

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЕ ПРИМЕНЕНИЕ

Новое
в жизни,
науке,
технике

Подписная
научно-
популярная
серия

Издается
ежемесячно
с 1988 г.

А.Ф. Дедков

Логическое программирование

АРХИТЕКТУРА ЭВМ

ПРОЛОГ

ПРОЛОГ

ЯЗЫКИ ПРОГРАМ-МИРОВАНИЯ

БАЗЫ ДАННЫХ

ПРОЛОГ

ПРОЛОГ

ИСКУССТВЕН-НЫЙ ИНТЕЛЛЕКТ

1988
9

Новое
в жизни,
науке,
технике

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА
И ЕЕ ПРИМЕНЕНИЕ

Подписная
научно-
популярная
серия

9/1988

Издается
ежемесячно
с 1988 г.

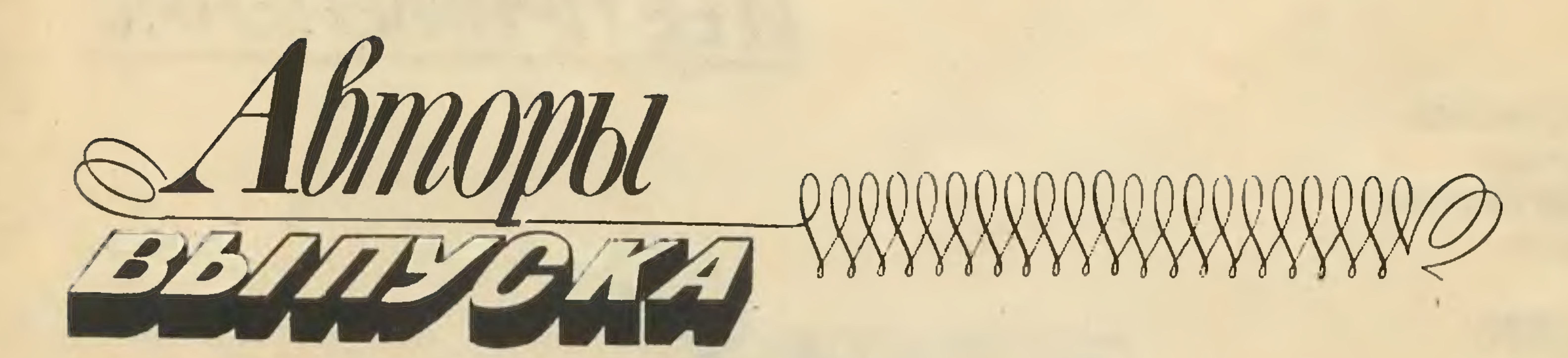
ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

В НОМЕРЕ:



Издательство
«Знание»
Москва
1988

ББК 32.973—01
Л 69



ДЕДКОВ Анатолий Федорович — кандидат технических наук, специалист в области программирования.

МАЛЫХИНА Мария Петровна — кандидат технических наук, доцент.

ЧАСТИКОВ Аркадий Петрович — кандидат технических наук, доцент.

БУСЛЕНКО Владимир Николаевич — кандидат технических наук, автор более 50 печатных работ в области имитационного моделирования сложных систем на ЭВМ.

МЕТАЛИДИ Александр Сократович — инженер-программист.

СЛОБОДЧУК Вячеслав Викторович — младший научный сотрудник института проблем управления. Занимается исследованием и разработкой локальных вычислительных сетей. Член Ассоциации пользователей БК.

ЛАДЫЧУК Игорь Николаевич — кандидат технических наук, доцент Днепропетровского сельскохозяйственного института.

БАРОНОВ Дмитрий Олегович, специалист в области бытовых компьютеров.

РЕДАКТОР Б. М. ВАСИЛЬЕВ



Что такое логическое программирование, каковы его основные идеи, что оно дает сейчас и обещает в будущем. Вот основные вопросы, на которые мы постараемся дать ответ в этой статье.

А. Ф. ДЕДКОВ

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

«В компьютер заложили исходные данные, и он мгновенно (через некоторое время) напечатал (вывел на экран дисплея) оптимальный вариант раскроя ткани (перевозки грузов, обмена квартиры и т. п.)». Фразы подобного рода иногда встречаются в газетных и журнальных статьях, посвященных успехам применения вычислительной техники в народном хозяйстве. Но в наш век всеобщих разговоров о глобальной компьютеризации и информатизации даже неспециалисты уже понимают наивность подобных высказываний восторженных журналистов. Мы знаем, что сколько ни закладывай данные в память компьютера, он никогда не решит самостоятельно ни одной задачи. Мы знаем также, что прежде, чем компьютер решит задачу, должна быть составлена программа ее решения. Мы знаем, что программирование на ЭВМ — это деятельность, требующая особого склада ума, высокой квалификации, большого опыта, максимальной сосредоточенности и внимательности. Мы знаем, что на составление даже простой программы потребуются часы, дни или даже недели, а сложной — годы.

Для профессиональных программистов — это привычное положение вещей, но для многих, впервые сталкивающихся с миром компьютеров, эта ситуация не только потрясает, но и озадачивает: «Неужели все это действительно необходимо? Неужели нельзя просто заложить данные в ЭВМ, поставив перед ней задачу, чтобы она сама нашла решение?» Профессиональные программисты могут, конечно, гордиться такой «потрясающей» оценкой их труда, но в глубине души и они не могут не нести сомнения в том, что современный способ программирования единственно возможный и правильный. Ведь если признать это, то придется признать и то, что армия программистов и дальше будет расти и расти, что многие имеющие практическую ценность задачи просто не будут решены из-за постоянного дефицита программистов. К тому же производительность их труда за последние 20 лет почти не выросла, что особенно огорчает на фоне тысячекратного роста за то же время производительности ЭВМ. Неужели же нет иного пути развития современного программирования? Оказывается, есть, хотя сейчас это пока еще не путь, это всего лишь тропинка. Называется она — логическое программирование.

Логическое программирование — это относительно новое перспективное направление современного программирования, возникшее первоначально в рамках работ по созданию систем искусственного интеллекта. Свое название логическое программирование получило от математической логики, лежащей в его основе. Следует уточнить: из всей математической логики используется только один из ее разделов, а именно исчисление предикатов первого порядка. Основная цель создания логического программирования — повышение «интеллектуальности» компьютеров. Однако роль логического программирования этим не ограничивается. В начале 80-х годов было осознано, что оно может стать связующим звеном, объединяющим в единую цепочку некоторые ранее слабосвязанные направления разработки и применения средств вычислительной техники:

совершенствование базовых принципов работы ЭВМ, так называемых архитектур ЭВМ;

развитие языков программирования;

создание систем искусственного интеллекта;

разработка так называемых языков запросов баз данных.

Причем, что очень важно, связь эта осу-

ществляется не через электрическое соединение плохо согласующихся между собой принципов и подходов, как это иногда делалось ранее и почти безрезультатно, а в рамках единой, простой и строгой концепции. Замечательно также и то, что сложность полученного таким способом объединения не только меньше суммы сложностей всех составных частей, но едва ли не меньше сложности каждой из них. Именно это послужило основанием для выбора логического программирования в качестве концептуальной основы широко известного японского проекта машин пятого поколения [1].

Наиболее полное и строгое выражение идеи логического программирования нашли в языке программирования ПРОЛОГ [2], что отражено даже и в названии этого языка (PROLOG — PROgramming in LOGic или ПРОЛОГ — ПРОграммирование ЛОГическое). Хотя ПРОЛОГ и не единственный язык программирования, на котором можно писать программы с использованием концепций логического программирования, но настолько известный, что с некоторых пор термины «логическое программирование» и «ПРОЛОГ» стали почти синонимами. Разработка языка ПРОЛОГ сделала логическое программирование практическим инструментом, доступным широким массам программистов. Последнее несомненно сыграло свою роль в том, что в настоящее время во всем мире наблюдается повышенный интерес к логическому программированию. Хотя примеров применения языка ПРОЛОГ для решения крупных практических задач пока еще немного, но и их достаточно, чтобы уверенно предсказать большое будущее этого языка.

Основные понятия

Каждый язык программирования, а рассказ о логическом программировании мы будем вести на примере конкретного языка — ПРОЛОГ, строится на двух фундаментах. Первый — это типы данных, с которыми разрешено оперировать в данном языке, их свойства и правила комбинирования друг с другом. Второй — это набор базовых конструкций языка с правилами их исполнения. От выбора этих основ языка зависит все. На первый взгляд кажется, что названные характеристики языка программирования должны полностью определяться конструкцией машины. Однако не так. Это справедливо только для так называемых языков низкого уровня, которые действительно разрешают оперировать только теми типами данных, которые могут храниться в памяти ЭВМ, и только теми операциями, которые она в состоянии выполнять. Как правило, эти типы данных столь элементарны: числа, символы (буквы, цифры, специальные знаки), операции над ними так примитивны (сложение, умножение, сравнение и т. п.), что составление даже простой программы на языке низкого уровня превращается в длительный кропотливый процесс и приводит к большому количеству ошибок, поиск и исправление которых обычно требуют даже большего времени, чем само написание программы. Кроме того, к недостатку программирования на языках низкого уровня относится и то, что программы оказываются привязанными к конкретной ЭВМ и не могут быть выполнены на машинах других типов.

Все эти недостатки языков низкого уровня заставили создать так называемые языки высокого уровня, ориентированные прежде всего не на конкретную машину, а на класс задач. Языки высокого уровня обладают более богатым набором типов данных и более удобным для программиста набором операций над ними. Отличается от языков низкого уровня и синтаксис языков высокого уровня, т. е. правила построения языковых конструкций. В настоящее время почти все программы разрабатываются на языках высокого уровня.

За последние 30 лет разработаны тысячи языков, однако в широком применении находится всего около десятка. Широко известны такие языки, как БЕЙСИК, ФОРТРАН, ПЛ / 1, ПАСКАЛЬ, СИ, МОДУЛА 2, АДА. Все эти языки относятся к классу так называемых процедурных языков. Значительно отличаясь друг от друга, они тем не менее

близки, так как имеют один и тот же «фундамент».

Каковы краеугольные камни этого фундамента? Хотя наша цель — логическое программирование, кратко рассмотреть основы процедурных языков необходимо, так как только тогда яснее станут отличия и достоинства логического подхода.

Основными типами данных, с которыми оперирует программист, работающий на процедурном языке, следующие. Числа, символы, последовательности символов (строки), логические значения «ложь» и «истина» и некоторые другие. Отдельные элементы данных можно объединять в конгломераты, называемые массивами и записями. Массив представляет собой объединение однотипных элементов, различаемых по номерам. Запись объединяет элементы данных различных типов, идентифицируемые по их именам. При необходимости могут строиться и объединения данных более сложной структуры: списки, деревья, графы и др.

Что же касается операционной части фундамента процедурных языков, то ее составляют следующие базовые операторы: оператор присваивания, условный оператор и оператор цикла.

Оператор присваивания изменяет текущее значение переменной на новое, причем старое значение бесследно исчезает.

Условный оператор позволяет в зависимости от текущего состояния вычислительного процесса выполнить то или иное действие.

Операторы цикла различных видов обеспечивают повторное выполнение некоторой части программы требуемое число раз с использованием в общем случае различных элементов данных.

Процедуры дают возможность выделить некоторую часть алгоритма решения задачи функционально законченной подзадачи, так что к ней можно обращаться многократно из разных мест программы. Процедуры являются мощным средством разбиения сложной задачи на части (структуризации программы), каждая из которых оказывается достаточно простой для понимания и обозримой.

Программы на языке программирования процедурного типа представляют собой, по существу, сложную и разветвленную совокупность приказов вида «сложи значения переменных A и B и запиши полученную сумму в переменную C» или «если A больше B, то дай переменной C то значение, которое имеет A, иначе то, которое имеет B». Такой способ программирования иногда называют императивным стилем программирования. При этом программист должен сначала продумать весь план решения задачи, а потом детализировать его до мельчайших подробностей. В этой «рабской» послушности машины приказам программиста одновременно и сила и слабость императивного стиля программирования на процедурных языках.

Кроме языков процедурного типа, существуют и другие классы языков: языки функционального и логического программирования. На первом классе мы не будем подробно останавливаться. Отметим только, что функциональное программирование применяется уже около 25 лет. Типичный и самый известный представитель этого класса — язык ЛИСП. На основных концепциях языка логического программирования мы остановимся более подробно, хотя и не будем давать полного описания языка ПРОЛОГ. Приводимые ниже сведения о языке не претендуют даже на роль краткого описания. Они нужны лишь для того, чтобы читатель мог с пониманием рассмотреть несколько приведенных ниже примеров и самостоятельно оценить их, вместо того чтобы доверять автору на слово.

Программы на языке ПРОЛОГ оперируют данными двух основных типов: числами и символами. Атом — это строка символов, обозначающая некоторый абстрактный объект. Атом считается неделимым элементом программы, в связи с чем и выбрано его название. Атом начинается со строчной буквы и состоит из букв, цифр и символа подчеркивания. Если атом должен содержать иные специальные знаки или начинаться с прописной бук-

вы, то он заключается в апострофы. Мы будем далее иногда использовать заключение атомов в кавычки для выделения их из окружающего текста.

Примеры атомов: машина, 'кто-то', иван.

Числа и атомы могут объединяться в конгломераты, называемые структурами и списками. Структура — это конструкция, состоящая из имени структуры и списка ее элементов, разделенных запятыми и заключенных в скобки. Элементами структур могут быть числа, атомы, переменные, другие структуры и списки. Количество элементов структуры фиксировано и не может изменяться при выполнении программы. Для тех, кто знаком с процедурными языками программирования, отметим, что структуры представляют собой нечто среднее между массивами и записями процедурных языков, так как типы элементов структуры могут быть различными, как у записей, а идентификация их производится по номерам, как у массивов.

Примеры структур: str (a, b, s, d)
имеет (иван, машина)

Списки представляют собой объединения элементов произвольных видов, разделенных запятыми и заключенных в квадратные скобки. Списки отличаются от структур тем, что не имеют имен, и тем, что количество их элементов может меняться при выполнении программы.

Примеры списков: [1, 3, 5, 7]
[красный, желтый, зеленый]

Элементами структур и списков могут быть так называемые переменные, обозначаемые именами, начинающимися с прописной буквы.

Примеры переменных: X, Y, Z

Основная операция, выполняемая над данными в языке ПРОЛОГ, это операция сопоставления (называемая часто также операцией унификации или согласования). Конечно, в языке есть и простейшие арифметические операции и операции сравнения, подобные аналогичным операциям процедурных языков, но операция сопоставления несет гораздо большую смысловую нагрузку и является одним из тех двух «китов», на которых стоит ПРОЛОГ.

Операция сопоставления определяется так:

а) число сопоставляется только с равным ему числом;

б) атом сопоставляется только с равным ему атомом;

в) переменная сопоставляется с чем угодно, при этом она получает в качестве значения то, с чем она сопоставляется;

г) структура сопоставляется с другой структурой, если они имеют одинаковые имена и количества аргументов, при условии, что все их элементы попарно сопоставимы.

Операция сопоставления может закончиться неудачно. В этом случае будем говорить, что ее операнды несопоставимы.

Примеры:

7 сопоставляется с 7,

«иван» сопоставляется с «иван»,

«иван» не сопоставляется с «петр»,

имеет (иван, машина) не сопоставляется с имеет (иван, телевизор),

имеет (иван, машина) сопоставляется с имеет (иван, X).

В последнем случае переменная X получает в качестве значения атом «машина». Правила сопоставления списков мы рассмотрим позднее.

Программа на языке ПРОЛОГ состоит из двух основных контрукций: фактов и правил. Факт — это структура, завершающаяся символом «точка». Факты представляют собой те данные, с которыми оперирует программа. Совокупность фактов, относящихся к некоторой задаче ПРОЛОГА, называется базой данных ПРОЛОГА. С помощью фактов описываются свойства объектов и отношения между ними. Факт, состоящий из структуры с одним элементом, обычно описывает некоторое свойство, например, факт «собака (бобик)» определяет, что объект «бобик» имеет свойство «быть собакой». Этот факт можно интерпретировать на естественном языке так: «Бобик есть собака» или более строго так: «Высказывание «Бобик есть собака» истинно». Факты с более чем одним элементом описывают взаимосвязи объектов (отношения между ними), например, факт «имеет (иван, машина)»

определяет, что объекты «иван» и «машина» находятся между собой в таком отношении, что второй является собственностью первого. Этот факт интерпретируется на русском языке так: «Иван имеет машину». Разумеется, ПРОЛОГ не понимает смысла имен, отношений и объектов, этот смысл приписывается им только в сознании программиста, разрабатывающего программу. В программе на ПРОЛОГЕ можно задать факты совершенно бессмысленные с точки зрения обыденного здравого смысла, например, «имеет (иван, юпитер)», т. е. «Иван является собственником планеты Юпитер». Для ПРОЛОГА этот факт ничем не хуже первого.

Другой составной частью программ на языке ПРОЛОГ являются правила. Правило есть конструкция вида:

A: — B, C, D,

где A, B, C, D — структуры. Структура A называется заголовком правила, а B, C, D — подцелями. Правило определяет истинность некоторого высказывания. Формальная запись правила интерпретируется так: «Высказывание A истинно, если одновременно истинны высказывания B, C и D». Вот пример правила:

```
имеет собаку (Некто): —
    имеет (Некто, Нечто),
    собака (Нечто).
```

Это правило определяет свойство объекта «иметь собаку» так: «Некто имеет собаку, если он имеет нечто, что является собакой».

По сути, выполнение программы на языке ПРОЛОГ представляет собой доказательство истинности некоторого логического утверждения в рамках данной совокупности фактов и правил. Алгоритм этого доказательства, называемый алгоритмом логического вывода, полностью определяет принципы выполнения программ на языке ПРОЛОГ и представляет собой того второго «кита», на котором стоит ПРОЛОГ.

Выполнение пролог-программы всегда начинается с ввода так называемого запроса, который представляет собой конструкцию вида

? — A, B, C.

Для выполнения некоторой совокупности подцелей, будь то запрос, или подцели, входящие в правило, ПРОЛОГ берет первую подцель и пытается доказать истинность этого утверждения (или, как мы будем говорить, выполнить подцель). Для этого просматриваются все факты и правила, составляющие программу, и ищется факт, сопоставимый с подцелью, или правило, заголовок которого сопоставляется с ней. Если такой факт найден, то истинность утверждения считается доказанной и берется следующая подцель. Если найдено правило, то ПРОЛОГ пытается по тем же правилам доказать истинность всех подцелей данного правила. Если при поиске обнаружено, что имеется несколько вариантов доказательства истинности подцели (т. е. есть несколько фактов или правил, сопоставимых с ней), то ПРОЛОГ автоматически отмечает так называемую точку возврата, т. е. запоминает альтернативные варианты решения. Если в какой-то момент выполнения программы очередная подцель не может быть выполнена, автоматически производится возврат к последней отмеченной точке возврата и ПРОЛОГ пытается найти другой вариант доказательства (или, что то же самое, другой путь выполнения программы). Автоматический перебор всех возможных вариантов решения является фундаментальным свойством языка ПРОЛОГ, которое часто оказывается крайне полезным и значительно сокращает объем текста программы. Кроме фактов и правил, определяемых программистом, в программе можно использовать так называемые встроенные предикаты, выполняющие всевозможные вспомогательные действия — вычисления, сравнение величин, ввод-вывод и др. Как видим, вместо десятков разнообразных операторов, из которых конструируется программа в процедурных языках, в языке ПРОЛОГ имеется всего один вид оператора — правило, что и определяет лаконичность языка. Вот, собственно, и все, что нам нужно знать о ПРОЛОГЕ, чтобы перейти к рассмотрению примеров, на которых мы продемонстрируем его свойства.

ПРОЛОГ и базы данных

В настоящее время наиболее популярны и многообещающи так называемые реляционные базы данных, в которых данные представляются в виде совокупности таблиц (отношений). Каждая строка таблицы описывает взаимоотношения нескольких объектов и называется кортежем. Кортеж состоит из отдельных элементов, называемых атрибутами. Как известно, такая простая структура данных тем не менее обеспечивает значительную гибкость и удобство реляционных языков запросов [4]. База данных ПРОЛОГА, состоящая из фактов, является не чем иным, как реляционной базой. Факт соответствует кортежу, его элементы — атрибутам, а совокупность фактов с одинаковым именем и одним и тем же количеством элементов — отношению. Запросы языка ПРОЛОГ представляют собой в этом случае запросы к базе данных, а сам ПРОЛОГ выступает в роли языка запросов. Рассмотрим пример. Разумеется, этот пример намеренно взят чрезвычайно простым, чтобы не затемнять свойства ПРОЛОГА спецификой конкретной сложной базы. Предположим, мы создали базу данных о группе людей и об имеющихся у них вещах. Объект «руб(N)» будет означать, что у данного индивидуума имеется N рублей. Допустим, база данных имеет такой вид:

имеет (иван, руб(1000)).
имеет (иван, машина).
имеет (иван, телевизор).
имеет (иван, магнитофон).
имеет (петр, руб(500)).
имеет (петр, телевизор),
имеет (петр, холодильник).
имеет (николай, руб (2000)).
имеет (николай, телевизор).

Рассмотрим различные виды запросов, которые можно вводить и получать на них ответы. Сначала будем формулировать запрос на русском языке, затем давать его в виде запроса на языке ПРОЛОГ, а затем приводить ответы, выдаваемые ПРОЛОГом.

1) Что имеет Петр?
Запрос: ? — имеет (петр, Вещь).
Ответ: Вещь=руб(500)
Вещь=телевизор
Вещь=холодильник.

Примечание: ПРОЛОГ выдает все возможные ответы на запрос за счет автоматического перебора всех вариантов решения. Заметим, что имя «петр» мы вводим со строчной буквы, так как это атом, а «Вещь» является переменной, поэтому начинается с прописной.

2) Кто имеет телевизор?
Запрос:
? — имеет (Человек, телевизор).
Ответ: Человек=иван
Человек=николай
Человек=петр

3) Кто имеет больше тысячи рублей?
Запрос:
? — имеет (Человек, руб (Сумма)).
Сумма>1000.
Ответ: Человек=николай.

4) Какие вещи имеет Иван, которых нет у Николая?
Запрос: ? — имеет (иван, Вещь),
не (имеет (николай, Вещь)).
Ответ: Вещь=руб (1000)
Вещь=машина
Вещь=магнитофон

Примечание. Заметим, что ПРОЛОГ выдал явно абсурдный ответ, что у Ивана есть 1000 рублей, которых нет у Николая, хотя мы видим, что денег у Николая даже больше, чем у Ивана. Это пример того, что ПРОЛОГ не понимает смысла запросов. Для него «руб(1000)» и «руб(2000)» это различные несопоставимые структуры. Если бы мы хотели исключить деньги из рассмотрения вообще, запрос надо было бы переформулировать так:

? — имеет (иван, Вещь),
Вещь /=руб(_),
не (имеет(николай, Вещь)).

Здесь операция «/=», как и «не» — это встроенные предикаты языка ПРОЛОГ. Символ «_» обозначает так называемую пустую переменную, значение которой нас не интересует.

5) Кто что имеет?
Запрос: ? — имеет (Кто, Что).
Ответ: Кто=иван Что=руб(1000)
Кто=иван Что=машина
Кто=иван Что=телевизор
Кто=иван Что=магнитофон
Кто=петр Что=руб(500) и т. д.

Добавим к нашей базе данных еще одно отношение «цена», связывающее продающиеся в магазине предметы с их стоимостью.

цена (машина, 7200).
цена (телевизор, 840).
цена (холодильник, 420).
цена (магнитофон, 350).
цена (видео, 1200).
цена (приемник, 130).
цена (часы, 50).

Теперь мы можем рассматривать запросы вида:

6) Может ли Петр купить видео?

Запрос:
? — имеет (петр, руб(Наличные)),
цена (видео, Цена),
Наличные > = Цена.

Ответ: Нет

Чтобы не вводить каждый раз такие сложные запросы, приведем правило, определяющее, что некто может купить вещь, если у него хватит денег и этой вещи у него нет.

может_купить (Некто, Вещь): —
имеет (Некто,руб(Наличные)),
Цена(Вещь, Цена),
Наличные > = Цена,
не (имеет (Некто, Вещь)).

Теперь можно вводить запросы такого вида:

7) Что может купить Николай?

Запрос:
?—может_купить (николай, Вещь).

Ответ: Вещь=холодильник
Вещь=магнитофон
Вещь=видео
Вещь=приемник
Вещь=часы

8) Что может купить Николай, чего не может купить (или в чем не нуждается) Иван?

Запрос: ? — может_купить (николай, Вещь),
не (может_купить (иван, вещь)).

Ответ: Вещь=видео

Мы можем ввести в рассмотрение и особ женского пола, при этом можно определить и отношение «является_женой», связывающее супружескую пару.

является_женой (анна, николай).
является_женой (мария, иван).
и т. п.

При этом если ввести предположение, что собственность мужа и жены общая, необходимо определить правило «у жены есть все, что есть у мужа». Для этого отношение «имеет» мы дополним правилом:

имеет (Женщина, Вещь): —
является_женой (Женщина, Мужчина). имеет (Мужчина, Вещь).

Заметим, что одно это правило заменяет множество фактов о том, что жены имеют все те вещи, которые имеют их мужья. При наличии этого правила ПРОЛОГ получает возможность как бы выводить новые факты из имеющихся. Например,

9) Имеет ли Мария машину?

Запрос: ?—имеет (мария, машина).

Ответ: Да

Примечание: ПРОЛОГ делает логический вывод, что если Мария является женой Ивана, а Иван имеет машину, то и Мария имеет машину.

Базы данных, программное обеспечение которых обладает способностью выводить новые данные из имеющихся, называются дедуктивными базами данных. Как видим, ПРОЛОГ также обладает таким свойством. В теоретических работах было показано, что возможности ПРОЛОГА как языка запросов к базе данных не уступают возможностям известных специализированных языков запросов реляционных баз. Причем, если последние относятся к специализированным языкам, не пригодным более ни на что, кроме своего прямого назначения, ПРОЛОГ реализует эти возможности вместе с возможностью использовать его как достаточно универсальный язык программирования. Важно то, что средства работы с базой в ПРОЛОГЕ являются не какой-либо «надстройкой» над языком программирования, а неотъемлемой частью базового механизма работы ПРОЛОГА. Рассмотрим далее ПРОЛОГ как язык программирования.

ПРОЛОГ
как язык программирования систем искусственного интеллекта

Для программистов, привыкших к процедурным языкам, ПРОЛОГ, несомненно, представляется весьма странным инструментом. Ведь в нем нет таких базовых конструкций, как оператор присваивания, условный оператор, оператор цикла, нет массивов и записей. Все это заменено операцией сопоставления структур и правилами логического вывода. Тем не менее оказывается, что на языке ПРОЛОГ можно программировать задачи весьма широкого класса. Хотя ПРОЛОГ ни в коей мере не является универсальным языком программирования, существует ряд классов задач, для решения которых он не применим. В первую очередь к таким классам относятся вычислительные задачи. Хотя ПРОЛОГ и обладает средствами выполнения вычислительных действий, но средства эти весьма слабы и носят преимущественно вспомогательный характер. Незачем применять ПРОЛОГ и для задач обработки данных, таких, как задачи экономического плана.

Для каких же классов задач пригоден ПРОЛОГ? Для таких, где главное значение имеют сложноструктурированные нечисловые данные, а также для таких, где важную роль должен играть поиск решения среди множества вариантов. Встроенный в ПРОЛОГ аппарат запоминания точек возврата и механизм автоматического возврата вычислительного процесса к ранее пройденному состоянию обеспечивают зачастую значительное сокращение объемов программ при переходе с процедурного языка на ПРОЛОГ.

Единственное, что, пожалуй, объединяет ПРОЛОГ с процедурными языками — это понятие процедуры. Процедурой в языке ПРОЛОГ называется совокупность правил с одним и тем же именем и одинаковым количеством параметров. Процедуры в процедурных языках всегда рассматриваются как приказы «сделать то-то и то-то над такими-то данными». Процедуры в языке ПРОЛОГ часто обладают таким свойством, что их можно рассматривать с двух точек зрения: с императивной, т. е. как приказы компьютеру выполнить необходимые действия в нужном порядке, и с декларативной точки зрения. В последнем случае они выступают в роли определения некоторого понятия. Декларативная трактовка процедур в ПРОЛОГЕ часто делает их более понятными человеку. Рассмотрим пример. Если в программе используются списки, то и в процедурных языках, и в ПРОЛОГЕ часто необходимо проверить, принадлежит ли некоторый объект данному списку. В процедурных языках эта процедура имеет следующий вид (мы используем условную запись на естественном языке):

Взять первый элемент списка, назвав его «текущим»;

Пока весь список не исчерпан, повторять следующие действия;

Если текущий элемент равен искомому, закончить цикл;

Сделать текущим следующий элемент списка;

Конец цикла.

В ПРОЛОГе формулировка понятия «быть элементом списка» делается так:

X является элементом списка Y, если X равен первому элементу списка, или X является элементом списка Z, полученного из Y отбрасыванием первого элемента. Запись соответствующей процедуры непосредственно на языке ПРОЛОГ выглядит так:

является_элементом (X, [Первый IZ]):
— X=Первый;
является_элементом (X,Z).

Операция «I» в ПРОЛОГЕ соответствует логической операции «ИЛИ». Операция «I» разделяет список на две части — первый элемент его и остальное. Если такая процедура введена в базу данных ПРОЛОГА, можно вводить запрос:

?—является_элементом (3,
[1, 3, 5, 7]).

Ответ: Да

В определении этой процедуры используется такой на первый взгляд странный прием, как определение некоторого понятия с помощью него же. Так, в определении понятия «является

элементом списка» мы пользуемся самим этим понятием так, как будто оно уже определено. В программировании такой прием называется рекурсией. В традиционных процедурных языках программирования рекурсия хотя и допускается, но используется редко, от случая к случаю. В языках же логического программирования рекурсия является основой всего. Она заменяет собой понятие цикла, как, например, в данной процедуре, и обеспечивает циклическое повторение некоторых действий над каждым из множества значений, заданного списком.

Необычной особенностью языка ПРОЛОГ является то, что в некоторых случаях можно задавать не все требуемые параметры процедуры. В этом случае ПРОЛОГ сам пытается подобрать подходящие значения. Например, если в запросе, содержащем вызов процедуры «является_элементом», вместо заданного значения указать переменную, то решением задачи будут все элементы списка.

?—является_элементом (N, [1, 3, 5, 7]).

Ответ:

N = 1
N = 3
N = 5
N = 7

Рассмотрим еще одну рекурсивную процедуру, подсчитывающую количество элементов списка (это количество программисты обычно называют длиной списка). Словесная формулировка ее будет следующей:

длина пустого списка равна нулю, длина непустого списка равна увеличенной на единицу длине списка, полученного из исходного отбрасыванием первого элемента.

Запись этой процедуры на ПРОЛОГе следующая:

```
длина ([   ], 0)
длина ([Первый | Остальные], Длина)
    : — длина (Остальные, Длина1),
    длина is Длина1+1.
```

Пример использования:

?—длина ([1, 3, 5, 7, 9], L).

Ответ: L = 5

Рассмотрим теперь, как же в языке ПРОЛОГ решаются задачи, требующие перебора многочисленных вариантов, часто встречающиеся в таких областях применения ЭВМ, как автоматизация проектирования, анализ фраз на естественном языке, построение экспертных систем и др.

К одной из разновидностей таких задач относится задача нахождения в графе пути, соединяющего заданные вершины и удовлетворяющего некоторым требованиям. К таким задачам относятся задачи выбора кратчайшего маршрута в транспортной сети, известная «задача коммивояжера» и др. Представителем этого же класса задач является известная детская задача, формулируемая так:

нарисовать домик, не отрывая карандаша от бумаги и не проводя два раза по одной и той же линии.

В более серьезной формулировке она читается так: «Обойти все ребра графа, не проходя два раза по одному ребру, так, чтобы в конце обхода вернуться в ту же точку. Пронумеруем вершины и ребра графа так, как показано на рис. 1. Нам необходимо задать структуру графа, указав, какие вершины соединены какими ребрами. Это сделаем с помощью отношения «ребро» следующего вида:

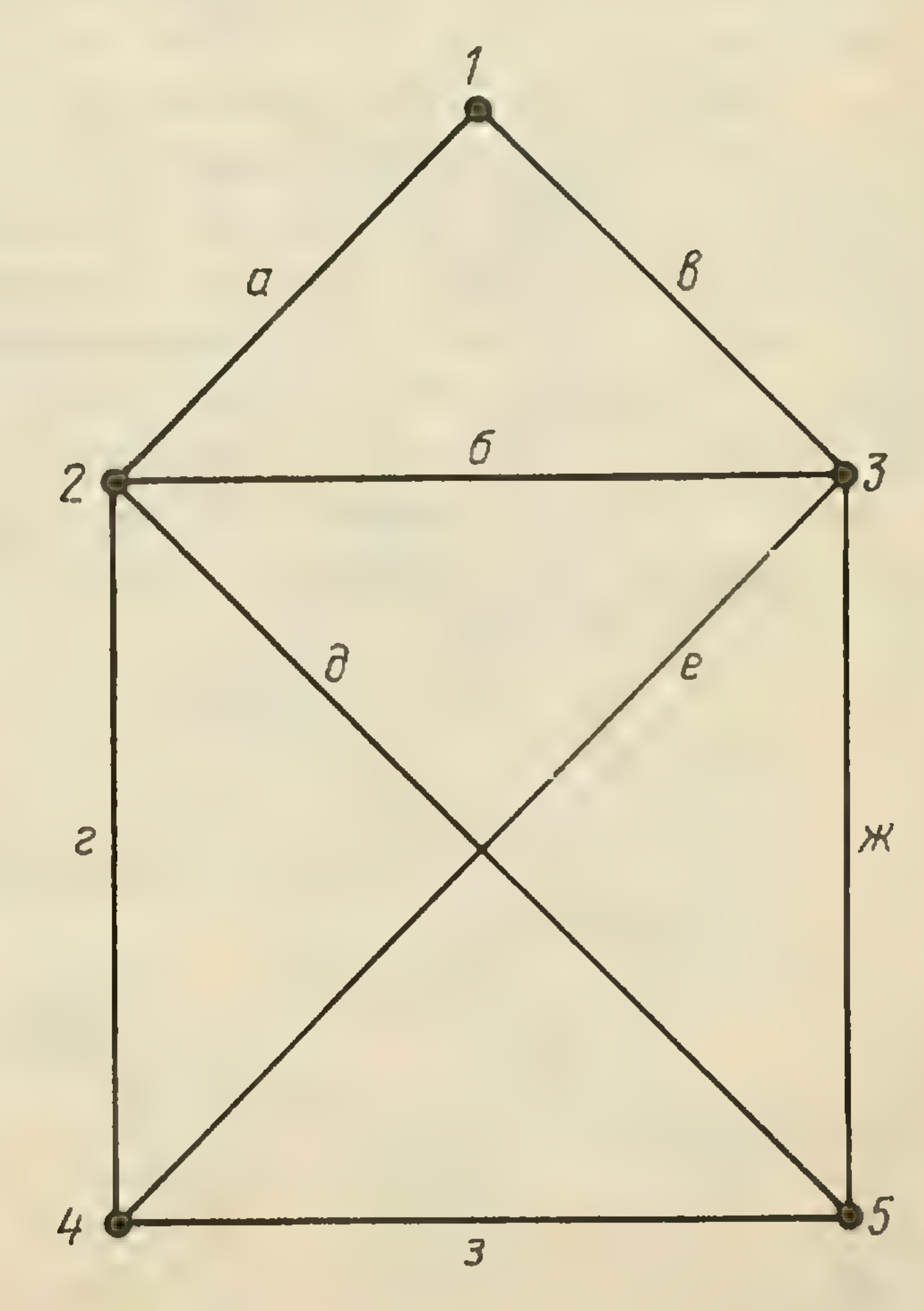


Рис. 1.

ребро (а, 1, 2). ребро (а, 2, 1).
ребро (б, 2, 3). ребро (б, 3, 2).
ребро (в, 1, 3). ребро (в, 3, 1).
... и т. п.

Заметим, что каждое ребро мы описываем дважды: как путь от вершины *i* к вершине *j*, и наоборот. Решение задачи описывается следующей процедурой «найти_путь»:

```
найти_путь (Текущая, Пройденные)
    : — длина (Пройденные, 8),
    write (Пройденные).
    найти_путь (Текущая, Пройденные): —
    ребро (Ребро, Текущая, Новая),
    не_является_элементом (Ребро, Пройденные),
    найти_путь (Новая,
    [Ребро I Пройденные).
```

Процедура имеет два параметра: «Текущая» вершина и список «Пройденные», хранящий номера пройденных в процессе поиска ребер. Используются две вспомогательные процедуры: «длина», рассмотренная ранее, и «не_является_элементом», аналогичная приведенной выше процедуре «является_элементом». Write — это так называемый встроенный предикат языка ПРОЛОГ, который выводит на дисплей значение своего аргумента, в данном случае списка. Первое правило этой процедуры гласит, что если найден путь, длина которого равна общему количеству ребер графа (в данном случае восьми), то искомый путь найден и его надо напечатать. Второе правило срабатывает, когда длина найденного отрезка пути меньше общего количества ребер, т. е. остались еще не пройденные ребра. В этом случае надо взять ребро, исходящее из текущей вершины и не включенное в список пройденных. Это ребро соединяет текущую вершину и вершину «Новая». Затем следует найти путь от новой вершины к конечной, добавив номер ребра к списку пройденных.

Как видим, эта процедура также имеет весьма простой вид, по крайней мере с первого взгляда. Но за этой простотой скрыт алгоритм логического вывода языка ПРОЛОГ, который и обеспечивает перебор всех возможных ребер, выходящих из каждой вершины, распознавание тупиковых ситуаций, возврат к тем вершинам, из которых выходят еще не опробованные пути и пр.

Для нахождения пути, по которому надо двигаться, чтобы нарисовать домик, начиная с четвертой вершины, следует ввести такой запрос:

?—найти_путь (4, [ ]).

Ответ: [з, ж, в, а, б, д, г, е]

Если мы введем запрос:

?—найти путь (1, [ ]).

Ответом будет: Нет, так как такого пути не существует.

Как видим, возможность решения задачи зависит от того, с какой именно вершины мы начинаем поиск. Можно воспользоваться тем, что в ПРОЛОГе разрешается иногда определять не все аргументы запроса, чтобы найти все множество вершин, которые могут служить начальными вершинами путей обхода графа. В ответ на запрос:

?—найти_путь (Начальная, [ ])

мы получим все множество возможных путей обхода и номера начальных вершин (для нашего домика такими вершинами будут четвертая и пятая).

К этому же классу задач принадлежит и известная задача о кенигсбергских мостах, впервые решенная великим математиком Эйлером. Постановка этой задачи такова.

Пройти по всем семи мостам г. Кенигсберга (XVIII в.), не проходя дважды по одному и тому же мосту (рис. 2).

Пронумеруем участки суши и обозначим каждую одной точкой (вершиной графа), тогда легко будет увидеть, что эта задача полностью аналогична рассмотренной выше задаче о построении домика. Отличается от нее только отношение «ребро», описывающее граф, и количество ребер. Набор фактов ПРОЛОГА, описывающий данный граф, теперь будет выглядеть так:

ребро (а, 1, 2). ребро (а, 2, 1).
ребро (б, 1, 2). ребро (б, 2, 1).
ребро (в, 1, 3). ребро (в, 3, 1).
ребро (г, 2, 3). ребро (г, 3, 2).
ребро (д, 2, 4). ребро (д, 4, 2).
ребро (е, 2, 4). ребро (е, 4, 2).
ребро (ж, 3, 4). ребро (ж, 4, 3).

Изменив в программе «найти_путь»

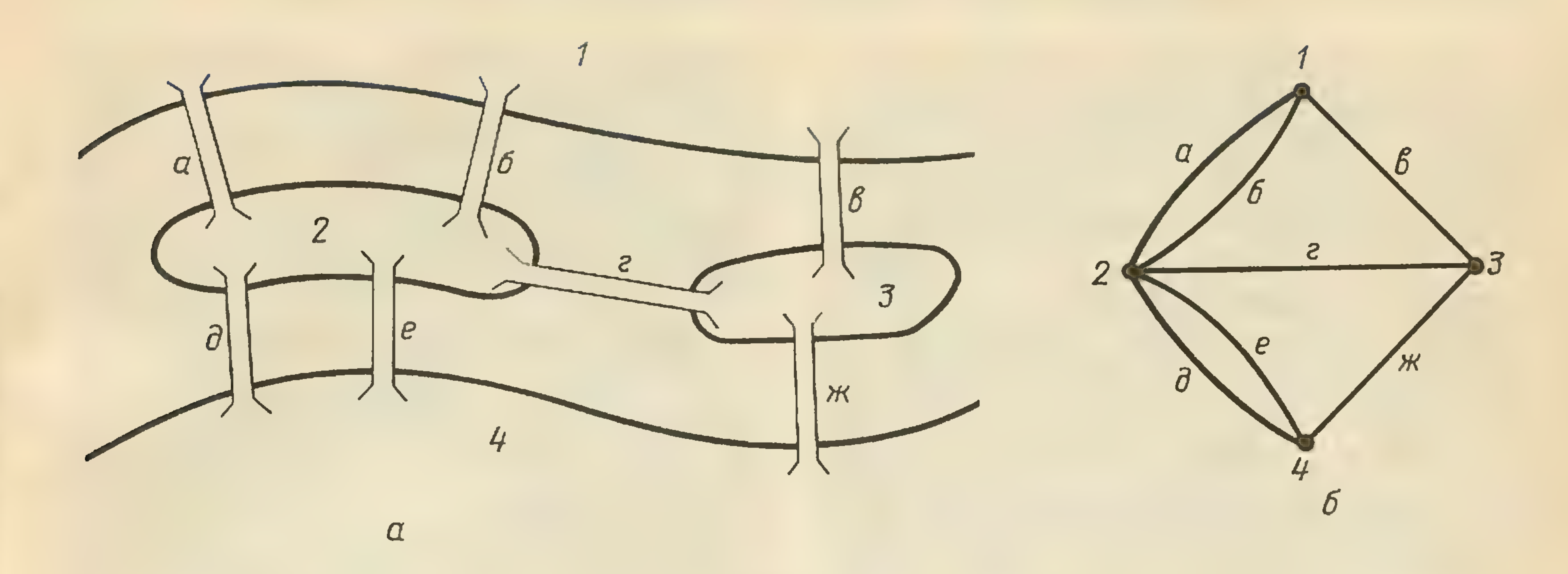


Рис. 2.

количество ребер с восьми на семь, введем запрос:

?—найти_путь (Начальная, [ ]).

Ответом ПРОЛОГа будет: Нет

К сожалению, наша программа подтверждает, что решения этой задачи не существует, что и было впервые строго доказано Эйлером в 1736 г.

Еще одна интересная особенность языка ПРОЛОГ: в процессе работы программы ее база данных может изменяться. Какие-то факты могут удаляться из нее, а другие добавляться. Причем это относится не только к фактам, но и к правилам. Это позволяет строить на языке ПРОЛОГ программы с элементами «самообучения». Например, к программе, рисующей домик, можно было бы добавить еще одну подцель для того, чтобы она запоминала в своей базе данных каждое найденное решение. Если впоследствии пользователь введет запрос на решение задачи с теми же исходными данными (т. е. для той же начальной вершины), ПРОЛОГ выдаст ему заранее заготовленное решение, уже не проводя поиска. Такой вариант программы представлен ниже.

Следует предупредить читателя, чтобы он не огорчался, если ему не во всех деталях понятно функционирование приводимых примеров на языке ПРОЛОГ. Приведенных выше сведений о работе ПРОЛОГа для этого просто недостаточно. Мы преследуем цель не изучить язык, а изложить в самом общем виде основные идеи логического программирования. Конкретные примеры приводятся исключительно по тому соображению, что лучше один раз увидеть, чем сто раз услышать. Можно сто раз услышать, что программы на ПРОЛОГе, решающие довольно сложные задачи, отличаются ясностью и краткостью, но лучше все-таки один раз увидеть эту краткость своими глазами. Если же наше изложение идей ПРОЛОГа подтолкнет читателя к более углубленному изучению языка для практического применения его в своей области деятельности, эти примеры, возможно, помогут ему в освоении языка.

Итак, вот вариант программы «найти_путь» с запоминанием найденных решений. Здесь asserta — встроенный предикат, добавляющий новое правило в базу данных:

найти_путь (Начальная, Текущая, Пройденные): —
длина (Пройденные, 8),
asserta (найти_путь (Начальная, Текущая, Пройденные): —
write (Пройденные)),
write (Пройденные).
найти_путь (Начальная, Текущая, Пройденные): —
ребро (Ребро, Текущая, Новая),
не_является_элементом (Ребро, Пройденные),
найти_путь (Начальная, Новая, [Ребро | Пройденные]).

Разумеется, ясно, что в нашей сверхпростой задаче о построении домика применение этого приема бес-

смысленно, так как существуют всего две допустимые вершины, с которых можно начинать построение. Однако представим себе, что мы разрабатываем на ПРОЛОГе справочную систему «Как проехать по Москве» для обслуживания пассажиров городского транспорта. Легко видеть, что эта задача того же класса: поиск кратчайшего пути в графе. Здесь множество различных запросов, огромно и запоминание готовых ответов на запросы вида: «Как проехать от ГУМа к универмагу «Московский»? или «... от универмага «Москва» к Казанскому вокзалу?» могло бы значительно сократить среднее время ожидания пользователем ответа системы. Понятно, что запоминание всех ответов на все запросы в данном случае быстро переполнило бы любой объем памяти системы. Видимо, здесь необходимо было бы учитывать частоту появления тех или иных запросов и удалять из базы данных ответы на редко встречающиеся запросы. ПРОЛОГ обеспечивает и эту возможность.

Почему мы уделили столько внимания задачам поиска путей в графах, решаемых ПРОЛОГом перебором всех вариантов. Дело в том, что этот класс задач тесно связан с задачами, возникающими при построении систем искусственного интеллекта. В теоретических работах, связанных с этой областью, было показано, что способностью к решению интеллектуальных задач обладают так называемые символические системы. Символическая система представляет собой множество символов, которые могут объединяться друг с другом в так называемые символические структуры. Некоторые символические структуры могут «быть интерпретированы», т. е. с ними могут быть связаны процедуры обработки данных. Решение интеллектуальных задач символическими структурами производится методом генерации различных решений и отбрасыванием неудовлетворительных, т. е. все тем же перебором. Легко видеть, что язык ПРОЛОГ вполне можно рассматривать как конкретную реализацию этих абстрактных символических систем. Символы — это атомы, символические структуры — структуры ПРОЛОГа, интерпретируемые структуры — правила, неинтерпретируемые структуры — факты, перебор вариантов — основа алгоритма логического вывода ПРОЛОГа. В силу этой близости ПРОЛОГ многими и рассматривается как перспективный язык для реализации систем искусственного интеллекта. Близкими свойствами обладает и язык ЛИСП, который более 25 лет применяется для построения интеллектуальных систем. Основное различие между ними в том, что ПРОЛОГ обладает встроенным механизмом логического вывода, а ЛИСП этого не имеет. В языке ЛИСП механизм логического вывода приходится разрабатывать самому программисту, что, естественно, усложняет для него задачу разработки программы. Однако такое преимущество ПРОЛОГа иногда может превращаться в недостаток. Это происходит тогда, когда встроенный в ПРОЛОГ алгоритм поиска по каким-либо причинам не подходит для решения данной конкретной задачи. Видимо, это и является одной из причин того, что ПРОЛОГ все еще не вытеснил ЛИСП из области построения систем искусственного интеллекта.

Рассматриваемое нами направление программирования называется логическим потому, что в его основе лежит аппарат математической логики. Естественно поэтому предположить, что оно должно хорошо подходить для решения тех классов задач, для которых математическая логика является наиболее подходящим инструментом. Это действительно так. Рассмотрим несколько примеров задач такого класса, взятых из сборника логических задач Р. М. Смаллиана [4]. Формулировки задач даются автором в легкой, подчас шутливой форме, но для решения их требуются некоторые способности к логичному мышлению и знание основ математической логики. Персонажи задач — это герои всемирно известных сказок Льюиса Кэролла об Алисе. В одной серии задач требуется определить, кто из жителей сказочной страны в своем уме, а кто — нет. При этом предполагается, что те, кто в своем уме, обо всем судят здраво, т. е.

мыслят только истинными утверждениями. Те же, кто не в своем уме, обо всем судят превратно. Если они думают, что утверждение *А* истинно, это значит, что на самом деле оно ложно, и наоборот. Вот одна из простых задач этой серии (номер 20):

«Король Пик думает, что Королева Пик думает, что она не в своем уме. В здравом ли уме Король и Королева Пик?»

Прежде чем продолжить чтение, попробуйте решить эту задачу самостоятельно и запомнить ход ваших рассуждений.

Решили? Тогда ход ваших мыслей, вероятно, был приблизительно таков. Предположим, что Король и Королева в здравом уме. Мы сразу приходим к противоречию, так как тогда Королева не может думать о себе заведомую ложь. Этот вариант отбрасываем. Предположим, что Король в своем уме, а Королева — нет. Этот вариант также не подходит из-за того, что тогда то, что думает Король — истинно, но Королева, находящаяся не в своем уме, не может судить о своем состоянии так здраво. Следующие два варианта, когда Король не в своем уме, а Королева либо в своем, либо нет, подходят. Независимо от состояния своего здоровья, Ее величество не может думать о себе так, как утверждается. Следовательно, это высказывание всегда ложно, а поскольку Король так думает, то он не в своем уме. Итак, Король не в своем уме, а относительно состояния здоровья Королевы мы ничего сказать не можем. Она может быть в любом состоянии.

Мы видим, что и человек решает эту задачу подобно ПРОЛОГу, перебором всех вариантов решения и отбрасыванием тех, которые приводят к противоречию. Как же решается эта задача с использованием ПРОЛОГа? Для этого надо сформулировать в виде фактов и правил ПРОЛОГа то, что должен знать человек, решающий задачу. Первое условие, гласящее, что каждое действующее в задаче лицо может быть либо в своем уме, либо нет, задается двумя фактами ПРОЛОГа:

участник (в_своем_уме).
участник (не_в_своем_уме).

Второе правило должно определить, что мы понимаем под состоянием «в своем_уме» и «не в своем уме». В словесной форме это правило выглядит так:

утверждение «X думает, что Y» истинно, если либо X в своем уме и Y истинно, либо X не в своем уме и Y ложно.

На языке ПРОЛОГ это правило записывается в следующем виде:

думает (X, Y): — (X = в_своем_уме, Y);
(X = не_в_своем_уме, не (Y)).

Напомним, что знак «;» обозначает логическую операцию «или», а не (Y) истинно, если Y ложно, и наоборот. Слова «в_своем_уме» и «не_в_своем_уме» для ПРОЛОГа не несут никакого смысла, они для него не более чем атомы, которые могут быть присвоены переменным в качестве значения. Для человека этих двух предварительных условий достаточно, чтобы решить задачу. Замечательно то, что и для ПРОЛОГа этих «знаний» оказывается достаточно. Если эти два факта и одно правило введены в базу данных ПРОЛОГа, мы можем вводить запрос, содержащий конкретное условие данной задачи:

?—участник (Король_Пик),
участник (Королева_Пик),
думает (Король_Пик,
думает (Королева_Пик,
Королева_Пик = не_в_своем_уме)).

В ответ на этот запрос ПРОЛОГ выводит два возможных варианта решения:

Король_Пик = не_в_своем_уме
Королева_Пик = в_своем_уме

Король_Пик = не_в_своем_уме
Королева_Пик = не_в_своем_уме

Мы видим, что ПРОЛОГ пришел к тем же выводам, что и мы, решая задачу в уме. Король всегда не в своем уме, а Королева может быть в любом состоянии. Примечательно, что мы вроде бы и не писали программу решения задачи, мы только сформулировали на ПРОЛОГе те предварительные знания, которыми должен обладать тот, кто решает задачу, и записали в другой форме словесную постановку задачи. Вот блестящий пример декла-

ративного стиля программирования. Мы получили решение, не строя алгоритма решения, а только сформулировав условие задачи.

Формулировку условий задачи можно средствами ПРОЛОГа сделать еще более похожей на исходную словесную постановку. Для этого можно воспользоваться таким средством языка ПРОЛОГ, как определение новых операций. Суть этой возможности в том, что, определив имя некоторой структуры как операцию, мы получаем возможность записывать в правилах и запросах

fx вместо f(x)

и xfy вместо f(x, y).

Определив имена «участник» и «думает» как операции, мы получаем возможность вводить запросы в таком виде:

```
?—участник Король_Пик,
участник Королева_Пик,
Король_Пик думает
Королева_Пик думает
Королева_Пик = не_в_своем_уме.
```

Интересно, что, определив предварительные условия задачи, мы тем самым сделали возможным решение не одной, а целой группы похожих задач. Вот еще одна задача (номер 23), решить которую в уме, пожалуй, потруднее, чем первую.

«Додо считает, что Лори считает, что Орленок Эд не в своем уме. Лори думает, что Додо не в своем уме, а Орленок думает, что Додо в здравом рассудке».

Для ПРОЛОГа же, как и следовало ожидать, решение такой задачи не представляет труда. Надо только сформулировать новое условие задачи в виде запроса:

```
? — участник Додо,
    участник Лори,
    участник Орленок_Эд,
    Додо думает Лори думает Орленок_Эд = не_в_своем_уме,
    Лори думает Додо = не_в_своем_уме,
    Орленок_Эд думает Додо = в_своем_уме.
```

Ответ ПРОЛОГа:

```
Додо = в_своем_уме
Лори = не_в_своем_уме
Орленок_Эд = в_своем_уме
```

И наконец, рассмотрим самую запутанную задачу из этой серии (номер 24). В ней действуют игральные карты. Вот ее условие:

«Тройка думает, что Туз не в своем уме. Четверка думает, что Тройка и Двойка оба не могут быть не в своем уме. Пятерка думает, что Туз и Четверка либо оба не в своем уме, либо оба в здравом рассудке. Шестерка думает, что Туз и Двойка оба в здравом уме. Семерка думает, что Пятерка не в своем уме. Что же касается Валета Червей, то он думает, что Шестерка и Семерка не оба не в своем уме».

Требуется определить, в своем ли уме Валет Червей. Вот формулировка запроса на ее решение, в котором операция «;» с помощью средств определения операций ПРОЛОГа заменена для удобочитаемости на «или»:

```
? — участник Двойка,
    участник Тройка,
    участник Четверка,
    участник Пятерка,
    участник Шестерка,
    участник Семерка,
    участник Валет,
    участник Туз,
Тройка думает Туз = не_в_своем_уме,
Четверка думает Тройка = в_своем_уме или Двойка = в_своем_уме,
Пятерка думает (Туз = в_своем_уме, Четверка = в_своем_уме) или
(Туз = не_в_своем_уме, Четверка = не_в_своем_уме),
Шестерка думает (Туз = в_своем_уме, Двойка = в_своем_уме),
Семерка думает Пятерка = не_в_своем_уме,
Валет думает Шестерка = в_своем_уме или Семерка = в_своем_уме.
```

Формулировка запроса достаточно сложна, но легко видеть, что она ничуть не сложнее условия задачи. В ответ на запрос ПРОЛОГ выдает нам четыре различных варианта решения:

```
Двойка = не_в_своем_уме
Тройка = не_в_своем_уме
```

Четверка = не_в_своем_уме
Пятерка = не_в_своем_уме
Шестерка = не_в_своем_уме
Семерка = в_своем_уме
Валет = в_своем_уме
Туз = в_своем_уме

Двойка = не_в_своем_уме
Тройка = в_своем_уме
Четверка = в_своем_уме
Пятерка = не_в_своем_уме
Шестерка = не_в_своем_уме
Семерка = в_своем_уме
Валет = в_своем_уме
Туз = не_в_своем_уме

Двойка = в_своем_уме
Тройка = не_в_своем_уме
Четверка = в_своем_уме
Пятерка = в_своем_уме
Шестерка = в_своем_уме
Семерка = не_в_своем_уме
Валет = в_своем_уме
Туз = в_своем_уме

Двойка = в_своем_уме
Тройка = в_своем_уме
Четверка = в_своем_уме
Пятерка = не_в_своем_уме
Шестерка = не_в_своем_уме
Семерка = в_своем_уме
Валет = в_своем_уме
Туз = не_в_своем_уме

ПРОЛОГ нашел четыре различных решения задачи, но поскольку в задаче спрашивается только, в своем ли уме Валет, решение на самом деле одно: „Валет в своем уме“, так как во всех вариантах решения он находится именно в этом состоянии.

Вопрос о том, можно ли решение этих задач считать проявлениями искусственного интеллекта машины, спорен. С одной стороны, никакой мистики здесь нет, и можно по этапам проследить, как ПРОЛОГ перебирает варианты и отбрасывает негодные. С другой стороны, человек, которому не откажешь в наличии интеллекта, причем не какого-то там «искусственного», решает эти задачи примерно так же, как и машина. И наконец, автор честно признается, что его уровня интеллекта не хватило для того, чтобы решить последнюю задачу. Еще честнее было бы сказать, что он и не пытался ее решить без помощи ПРОЛОГа ввиду ее сложности. А ведь для ПРОЛОГа это далеко не вершина его «интеллектуальных» возможностей, это тривиальная задача, решаемая почти мгновенно.

Оставив в стороне вопрос о том, являются ли эти задачи примерами интеллектуальных задач, отметим только, что в них ПРОЛОГ явственно демонстрирует возможности решения задач без детального программирования алгоритмов их решения. Заметим, что если программисту, работающему на традиционных процедурных языках программирования, предложить написать программу, решающую этот класс логических головоломок, причем с тем условием, чтобы исходное описание задачи вводилось в не менее естественном, чем в ПРОЛОГе, виде, то написание и отладка такой программы займут у него несколько дней, а то и недель, и общий объем ее будет не менее нескольких сот строк, а скорее всего, превысит тысячу (вместо трех строк на ПРОЛОГе). Чтобы не создавать ложного представления о мощности методов логического программирования вообще и ПРОЛОГа как языка логического программирования в частности, заметим, что такое соотношение объемов программ на ПРОЛОГе и на более традиционных языках программирования наблюдается только для некоторых классов задач. Для других же классов преимущества ПРОЛОГа более скромны, для иных же их и вовсе нет.

Рассмотрим еще одну задачу. В отличие от предыдущих логических задач эта задача принадлежит к классу алгоритмических головоломок, в которых требуется найти алгоритм, т. е. последовательность действий, приводящую к решению. Формулировка задачи такова. Даны три сосуда емкостью 10, 7 и 3 литра. В первом сосуде содержится 10 литров молока. Требуется разделить молоко поровну, пользуясь только этими тремя сосудами. Мы видим, что здесь нужно найти именно алгоритм, т. е. последовательность переливаний из одного сосуда в другой. Задача для любого программиста

необычная. Программист, как правило, составляет алгоритм, а машина только выполняет его, здесь же требуется, чтобы машина нашла алгоритм. Попробуем сформулировать условия задачи на языке ПРОЛОГ, как мы это уже делали ранее.

Информацию о емкости сосудов представим фактом

объемы (10, 7, 3).

Основная и единственная операция, которую мы можем выполнять над сосудами, — переливание из одного в другой. Ясно, что перелить можно только в неполный сосуд. Количество молока, которое переливается, определяется так. Если в первом сосуде молока больше, чем свободного места во втором, то переливается количество молока, равное свободному объему второго сосуда. В противном случае из первого сосуда выливается все молоко. Иными словами, количество переливаемого молока равно минимуму из его количества в первом сосуде и свободного объема второго. Что такое минимум, ПРОЛОГ «не знает», поэтому нам придется выразить это понятие в виде правила:

минимум двух чисел A и B равен A, если A меньше B, и равен B, если B меньше или равно A.

На ПРОЛОГе это записывается так:

минимум (A, B, A): — A < B.
минимум (A, B, B): — B = < A.

Теперь мы можем сформулировать правило переливания из одного сосуда в другой. Чтобы перелить из одного сосуда в другой, надо, чтобы в первом сосуде было какое-то количество молока, а во втором сосуде — свободное место. Переливаемое молоко по количеству равно минимуму из этих величин. После переливания молока в первом сосуде станет меньше на перелитое количество, а во втором увеличится на столько же. Количество молока в третьем сосуде при таком переливании не изменится. После выполнения переливания следует попытаться найти другую возможность переливания из нового состояния так, чтобы в конце концов прийти к такому состоянию, когда в двух первых сосудах молока станет по 5 литров.

Текущее состояние сосудов описывается тройкой чисел, определяющих, сколько молока в каждом. Учитывая это, запишем правило переливания на ПРОЛОГе в следующем виде (предполагая, что мы переливаем из первого сосуда во второй):

перелить (В_первом, Во_втором, В_третьем): —
объемы (Объем 1, Объем 2, Объем 3),
В_первом > 0,
Свободно is Объем 2 — Во_втором,
Свободно > 0,
минимум (В_первом, Свободно, Количество),
Стало_в_первом is В_первом – Количество,
Стало_во_втором is Во_втором + Количество,
перелить (Стало_в_первом, Стало_во_втором, В_третьем).

Как видим, это правило опять есть не что иное, как переформулировка ранее словесно выраженного правила на язык логического программирования. Затем следует сформулировать еще пять полностью аналогичных правил переливания из первого сосуда в третий, из второго в первый и т. п. Условие завершения этого процесса переливания мы сформулируем фактом

перелить (5, 5, 0).

Если при очередной попытке выполнения подцели «перелить» ПРОЛОГ обнаружит, что в двух первых сосудах стало ровно по 5 литров, вычисления закончатся. Переформулировка задачи с русского языка на язык ПРОЛОГ закончена, и как всегда вместе с этим закончено и построение программы, решающей задачу. Введем теперь запрос

? — перелить (10, 0, 0).

В нем мы указываем начальное состояние сосудов, в первом — 10 литров, в остальных — ничего. Однако выполнения введенного запроса нам придется ждать очень долго, так как оно никогда не закончится. Дело в том, что мы забыли сформулировать еще одно правило.

Если после некоторого переливания мы придем к состоянию, в котором сосуды уже однажды были, то такое переливание делать не надо.

В нашем же случае в строгом

соответствии с заданными правилами ПРОЛОГ будет переливать из первого сосуда во второй, затем из второго обратно в первый, снова из первого во второй и так до бесконечности. Придется нам несколько усложнить правило переливания. Введем в число параметров правила еще один — список всех пройденных состояний, подобно тому как это делалось при решении задачи о построении домика. Каждое новое состояние будем проверять на принадлежность этому списку. Теперь правило «перелить» будет выглядеть так:

перелить (В_первом, Во_втором, В_третьем, Пройденные): —
объемы (Объем 1, Объем 2, Объем 3),
В_первом > 0,
Свободно is Объем 2 — Во_втором,
Свободно > 0,
минимум (В_первом, Свободно, Количество),
Стало_в_первом is В_первом — Количество,
Стало_во_втором is Во_втором + Количество,
Новое=сост (Стало_в_первом, Стало_во_втором, В_третьем),
не_является_элементом (Новое, Пройденные),
перелить (Стало_в_первом, Стало_во_втором, В_третьем, [Новое I Пройденные]).

Факт «перелить (5, 5, 0)» надо также изменить, так как мы ввели дополнительный параметр. Теперь он будет записываться так:

перелить (5, 5, 0, Пройденные).

Вот теперь в ответ на запрос

? — перелить (10, 0, 0, [ ])

ПРОЛОГ выдаст ответ, но ответ обескураживающий: «Да». ПРОЛОГ доказал, что существует последовательность переливаний, приводящая из исходного состояния в конечное. Для того чтобы вывести эту последовательность, надо предпринять специальные действия, а именно заменить факт

перелить (5, 5, 0, Пройденные)

на правило

перелить (5, 5, 0, Пройденные): —
write (Пройденные).

Теперь наша программа полностью завершена, и при вводе запроса выдаст все возможные алгоритмы переливания. Всего, как выясняется, существует двадцать алгоритмов, в каждый из которых входит от десяти до двадцати переливаний. Если мы будем записывать каждое состояние тройкой чисел, то кратчайшее решение будет выглядеть так:

10 0 0
3 7 0
3 4 3
6 4 0
6 1 3
9 1 0
9 0 1
2 7 1
2 5 3
5 5 0

Из этого примера можно сделать вывод: говоря о том, что для решения задачи на языке ПРОЛОГ надо только переформулировать ее условия на строгом языке, мы отчасти вводим читателя в заблуждение. Такая переформулировка — процесс далеко не тривиальный, при этом могут быть сделаны ошибки (без которых, как известно, не обходится построение любой программы на любом языке). Выявление ошибок может потребовать многочисленных запусков программы на выполнение, т. е. того, что в программировании называется отладкой программы. Для облегчения отладки в ПРОЛОГе предусмотрены специальные средства, позволяющие программисту прослеживать выполнение программы шаг за шагом. Таким образом, деятельность по решению сложных задач с помощью логического программирования требует от программиста значительных интеллектуальных усилий. Логическое программирование — не панацея от всех бед, а все-таки именно методика программирования. Иное дело, что усилия, затрачиваемые на разработку программы в рамках логического программирования, могут быть во много раз меньше, чем в процедурном программировании.

Пролог и ЭВМ будущего

Мир вычислительной техники стремительно развивается. В течение жизни одного поколения он изменился неузнаваемо. Длинные шеренги металлических шкафов, битком набитых сначала электронными лампами, а затем транзисторами, сменили компьютеры, умещающиеся на письменном столе или даже в небольшом чемоданчике. По своим параметрам они зачастую превосходят гигантские машины прошлого. Это не может не вызывать удивления даже у тех, кто вплотную в течение всей жизни сталкивался с ЭВМ и видел своими глазами их эволюцию. Так, центральный процессор отечественной персональной ЭВМ, с помощью которой пишутся эти строки, представляет собой маленький кристаллик кремния, размер которого меньше копеечной монеты. Вместе с тем его быстродействие в сотни раз превышает быстродействие тех ЭВМ, с которыми автор 20 лет назад начинал свою деятельность в области программирования. Объем ее оперативной памяти в десятки раз больше, он достаточен для хранения в нем текстов пяти книг, равных по объему этой. Однако, несмотря на интенсивное развитие технологии производства ЭВМ и успехи микроминиатюризации, можно сказать, что компьютеры за последние 30 лет практически не изменились. Изменился их внешний вид, изменились размеры и технические параметры, т. е. количественные пока-



Название свое «АЛГОЛ» (ALGOL) получил от сокращения слов ALGOritmic Language, что в переводе означает «алгоритмический язык», хотя вначале он назывался ИАЛ (IAL — The International Algebraic Language).

Работы по созданию алгоритмического языка были начаты в 1955—1956 гг. в США Ассоциацией по вычислительной технике (АСМ) и в Европе немецким Обществом прикладной математики и механики (GAMM). В 1958 г. АСМ и GAMM подготовили совместный предварительный отчет о языке, который получил название АЛГОЛ-58.

На конференции в Париже в январе 1960 г. после широкого обсуждения представители семи стран — США, ФРГ, Англии, Франции, Дании, Нидерландов и Швейцарии утвердили улучшенную версию языка, названную АЛГОЛ-60. Двумя годами позже на конференции в Риме были сформулированы поправки и дополнения к описанию языка, но свое название (АЛГОЛ-60) он не изменил. Большие заслуги в создании, развитии и пропаганде АЛГОЛа принадлежат Д. Бэкусу, П. Науру и другим ученым.

Авторы языка в одном из первых отчетов указали такие цели его создания. Во-первых, новый язык должен быть близок к обычному математическому языку, во-вторых, он должен быть удобен для использования описаний алгоритмов, в-третьих, язык должен быть механически переводимым на машинные языки. Главным образом АЛГОЛ предназначался для решения научных и инженерных задач.

Какие же отличительные черты характеризуют АЛГОЛ?

1. АЛГОЛ во многих отношениях рассматривается как улучшение такого языка, как ФОРТРАН.

2. Как язык программирования он был определен независимо от какой-либо машинной реализации.

3. Ученые получили удобное средство для описания алгоритмов и обмена ими между пользователями.

4. При создании АЛГОЛа был продемонстрирован новый стиль языкового описания, это первый язык, имеющий строго определенный синтаксис. Идея формально-синтаксической системы записи для языков программирования была введена Бэкусом в 1959 г. (систему стали называть БНФ-Бэкуса нормальная форма).

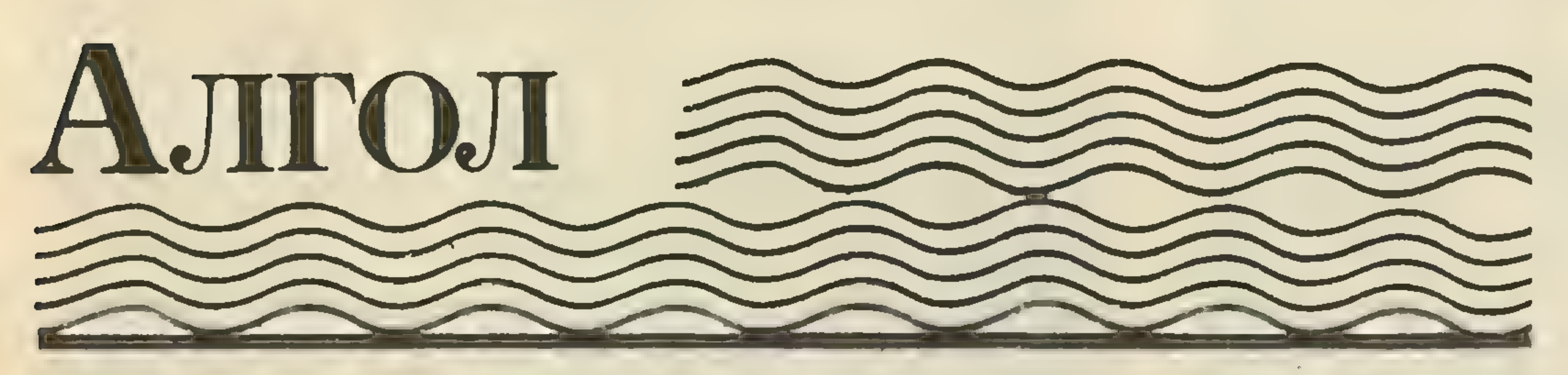
5. Появление АЛГОЛа привнесло новые идеи в разработку языков программирования, наиболее существенные из них относятся к блокам и процедурам. АЛГОЛ стал первым языком программирования с блочной структурой. АЛГОЛьные программы представляют собой независимые друг от друга блоки операторов. Имена переменных, процедуры, выполняемые и невыполняемые операторы относятся только к конкретному блоку. Блоки могут быть вложенными друг в друга.

АЛГОЛ оказал значительное влияние на последующее развитие языков высокого уровня, причем выделяют три направления этого влияния. Первое характеризуется расширением языка в области структур и операций (пример — язык СИМУЛА), второе — большей степенью формализации и обобщения языка и его описания (пример — язык АЛГОЛ-68). В третьем — сохранились стиль и размеры описания АЛГОЛа, но введены новые концепции, которые еще в большей степени

затели, качественных же изменений не произошло.

Компьютеры по-прежнему строятся в соответствии с так называемой моделью фон-Неймана, предложенной более 40 лет назад. Основная идея этой модели в том, что память машины устроена как линейная последовательность ячеек, каждая из которых характеризуется своим номером (адресом). В памяти хранятся как сами программы, так и обрабатываемые ими данные. Машина выполняет программу последовательно, команда за командой в том порядке, как они расположены в памяти, за исключением случая, когда очередная команда явно предписывает произвести переход на другой участок памяти. Этой модели следуют и процедурные языки программирования. Именно поэтому они в наилучшей степени подходят для построения эффективных программ на ЭВМ с такой конструкцией, или, как говорят, с фон-неймановской архитектурой.

Конечно, в процессе развития вычислительной техники делались попытки создать ЭВМ другой архитектуры. Часто эти попытки стимулировались новыми идеями, найденными в процессе развития языков программирования. Так, например, появление и интенсивное использование языка ЛИСП (особенно в работах по созданию систем искусственного интеллекта) привело к созданию особого класса компьютеров, так называемых Лисп-машин. Эти машины, архитектура кото-



сочетают простоту и всеобщность (пример — язык ПАСКАЛЬ). На основе АЛГОЛа были разработаны многие диалекты, причем некоторые из них (МАД и ДЖОВИАЛ) в развитии отклонились от своего истока и стали независимыми языками.

В СССР в 1965 г. на базе АЛГОЛ-60 и средств языка КОБОЛ был создан язык АЛГЭК, ориентированный преимущественно для программирования экономических задач. Компилятор с этого языка впервые реализован на ЭВМ «Минск-22». В 1964—1966 гг. также на базе АЛГОЛа был разработан язык АЛГЭМ, предусматривающий собой сокращенный вариант АЛГОЛа-60 с введением величин типа «строчный», строчными выражениями и функциями, составными переменными и массивами. Компилятор с языка реализован также на ЭВМ «Минск-22».

В 1967 г. группой ГАМС (Группа по Автоматизации программирования для Машин Среднего типа), созданной Комиссией многостороннего сотрудничества академий наук социалистических стран, на базе АЛГОЛа-60 была разработана версия АЛГАМС. В дальнейшем эта версия была реализована на ЕС ЭВМ, а в 1976 г. был принят стандарт на язык АЛГАМС(ГОСТ 21551—76).

АЛГОЛ оказал влияние на проектирование архитектуры многих вычислительных систем, в частности, серии компьютеров В5000. К сожалению, АЛГОЛ имеет несколько слабых мест, и главное из них — бедные средства ввода-вывода.

Язык АЛГОЛ-68 разработан в 1968 г. (в переработанном виде — в 1975 г.) с развитыми средствами ввода-вывода, но из-за конкуренции с другими языками он не получил широкого применения.

Анализируя пройденный АЛГОЛом путь, необходимо заметить, что, несмотря на повсеместное использование языка в европейских странах, в США он не нашел широкого распространения. Объясняется это прежде всего коммерческими соображениями: пользователи, затратив значительные средства на ФОРТРАНовское программное обеспечение (незадолго до появления АЛГОЛа), стали противниками каких-либо изменений в своей программистской практике. Этому в немалой степени также «способствовали» успехи и признание компьютеров фирмы IBM, взрастившей ФОРТРАН.

Время неумолимо, и даже в тех странах, где АЛГОЛ в течение многих лет был основным языком программирования, в настоящее время в связи с появлением новых более совершенных языков интерес к нему стал угасать, но прошлые его заслуги бесспорны.

Литература

1. Наур П. (ред.) — Сообщение об алгоритмическом языке АЛГОЛ-60 / / Вычислительная математика и математическая физика. — 1961. — № 2.
2. Королев М. А. и др. Сообщение об алгоритмическом языке для экономических расчетов АЛГЭК / / Кибернетика. — 1966. — № 2.
3. Лавров С. С. Введение в программирование. — М.: Наука, 1973.
4. Осипов Л. А. Программирование на алгоритмических языках. — М.: Знание, 1981.
5. Хелмс Г. Л. Языки программирования / Пер. с англ. — М.: Радио и связь, 1985.

М. МАЛЫХИНА,
А. ЧАСТИКОВ

рых разработана с ориентацией именно на язык ЛИСП, позволяют значительно повысить быстродействие программ, написанных на этом языке.

Логично предположить, что появление такого нового направления, как логическое программирование, вызовет к жизни новый класс архитектур ЭВМ. Действительно, такие машины уже появляются. Они называются обычно Пролог-машинами или машинами логического вывода. Зачем нужны эти новые разработки, ведь и на привычных машинах с архитектурой фон-Неймана Пролог-системы успешно разрабатываются и применяются? Да, конечно, но при этом разработчикам их приходится выражать алгоритмы унификации и логического вывода, присущие ПРОЛОГу, на языке машин, ориентированных на языки процедурного типа. Это, во-первых, достаточно сложно, а во-вторых, что более важно, не эффективно. Приблизив концепции построения машины к концепциям построения того языка, на котором производится программирование, мы сможем при тех же затратах увеличить быстродействие в десятки, а может быть, и в сотни раз.

А критерий быстродействия во все времена был одним из основных при сравнении различных видов ЭВМ. Для логического же программирования, где упрощение процесса программирования достигается, как читатель мог уже убедиться, за счет расточительного использования ресурсов ЭВМ, уходящих на перебор и отбрасывание сотен, тысяч, а то и миллионов вариантов, быстродействие играет особо важную роль. В чем же принципиальное отличие Пролог-машин от машин с архитектурой фон-Неймана? Их два.

Первое — структура памяти. В машинах традиционной архитектуры, как говорилось, каждая ячейка памяти имеет свой адрес. Чтобы процессор смог прочитать данные из памяти, он должен знать адрес того участка памяти, где эти данные «лежат». В языке ПРОЛОГ подход принципиально иной. Выборка данных из памяти (базы данных ПРОЛОГа) ведется на основе так называемого ассоциативного поиска. При этом должен быть задан некоторый «шаблон», определяющий структурную упорядоченность необходимых элементов данных, где именно они расположены в памяти, совершенно безразлично. Так, например, чтобы получить данные об объемах сосудов в последней рассмотренной нами задаче, нам не нужно знать, где эти данные записаны. Важно было знать другое, что эта информация сгруппирована в структуру с именем «объемы», содержащую три элемента данных. При выполнении подцели «объемы (А, В, С)» ПРОЛОГ автоматически обеспечит поиск нужной структуры в памяти ЭВМ.

При реализации языка ПРОЛОГ на машинах традиционной архитектуры ассоциативный поиск данных имитируется с помощью связывания элементов данных в структуры, называемые списками, где каждый элемент списка хранит адрес следующего элемента. Процедура поиска в этом случае может требовать выполнения достаточно сложной программы и многократного обращения к памяти. Понятно, что если бы ассоциативный поиск мог выполняться непосредственно самой памятью машины, он был бы существенно быстрее. Таким образом, мы видим, что в Пролог-машинах структура памяти должна быть иной, чем у машин фон-Неймана. Вместо выборки данных по адресам должен быть обеспечен ассоциативный поиск данных.

Второе отличие Пролог-машин от машин традиционной архитектуры — в принципах выполнения программ. В последних основой выражения логики программы являются операции сравнения и перехода. В языках программирования высокого уровня им соответствуют так называемые условные операторы. В ПРОЛОГе же, как мы видели, операции сравнения хотя и существуют (например, подцель «Свободно > 0» в последней задаче), но играют второстепенную роль. Основной же аппарат, обеспечивающий ветвление вычислительного процесса, — это создание точек возврата и операция возврата к ранее пройденному состоянию, если процесс вычислений зашел в «тупик». При реализации языка ПРОЛОГ на

машинах фон-Неймана эти операции тоже приходится имитировать с помощью неестественных для этой цели средств, что ухудшает быстродействие. Если бы эти операции выполнялись самим процессором, т. е., как говорят, были «аппаратно реализованы», время выполнения программ на языке ПРОЛОГ уменьшилось бы существенно.

Сказанное подтверждается и имеющимся уже, пока небольшим, опытом создания и эксплуатации Пролог-машин. Так, например, быстродействие персональной Пролог-машины PSI, разработанной в рамках работ по созданию ЭВМ пятого поколения в Японии, превышает быстродействие Пролог-системы, реализованной на большой ЭВМ IBM-3033, имеющей традиционную архитектуру [5]. Объявлено, что быстродействие следующей модели PSI-II будет еще в 3 раза больше [6].

Однако приближение архитектуры машины к требованиям языка программирования не единственный способ повышения быстродействия, который обеспечивает следование принципам логического программирования. Есть и еще одно обстоятельство. Как известно, увеличение быстродействия ЭВМ за счет совершенствования электронных схем имеет предел. Предел этот установлен таким фундаментальным ограничением, как скорость света. Никакой сигнал не может передаваться быстрее 300 тыс. километров в секунду. На первый взгляд кажется, что столь невообразимо огромное значение не может иметь никакого отношения к процессам, происходящим в машине. Однако это не так. Процессы, происходящие в современной супер-ЭВМ, выполняющей около 1 млрд. операций в секунду, лежат в пикасекундном диапазоне (1 пс — это тысячемиллиардная доля секунды). Легко подсчитать, что за 1 пс свет проходит менее 1 мм (!). Это значит, что электрический сигнал пройдет по кремниевому кристаллу из конца в конец примерно за 20 пс. Кроме того, следует учитывать, что электрический сигнал в проводнике распространяется все-таки медленнее, чем кванты света в вакууме. Таким образом, современная электроника уже сейчас, на наших глазах, подошла к „световому барьеру", оставив далеко позади космическую технику, околосветовые фотонные звездолеты которой все еще существуют только в книгах фантастов.

Каков же выход? Что поможет преодолеть этот барьер? Ответ давно известен. Это параллельная обработка данных. Идея тривиально проста. Если скорость света мешает нам создать ЭВМ с быстродействием 10 млрд. операций в секунду, сделаем десять ЭВМ по 1 млрд. каждая и заставим их вместе решать одну задачу. Попытки создания параллельных ЭВМ более или менее успешно ведутся уже свыше 25 лет. Более того, каждая мощная ЭВМ сейчас обладает так называемым внутренним параллелизмом, состоящим в том, что параллельно выполняются отдельные микроэтапы одной машинной команды.

Казалось бы, все хорошо. Но нет! Сложность в пустяке. Как заставить десять машин решать одну задачу. Разрабатываются специальные языки программирования, предусматривающие возможность распараллеленного выполнения программ. Создаются компиляторы, автоматически обнаруживающие в программах участки, которые можно выполнять одновременно. Но за исключением некоторых достаточно узких классов задач, например задач линейной алгебры или решения дифференциальных уравнений, общего универсального решения не найдено. Проблема параллельных вычислений остается в большой степени проблемой.

Чем же здесь может помочь логическое программирование? Как мы видели, процесс выполнения программы на языке ПРОЛОГ предусматривает перебор массы возможных путей решения задачи. Если изобразить эти пути графически, мы получим нечто вроде дерева, растущего корнями вверх. В основном стволе лежит исходный запрос, в концах некоторых веточек — решения. Большинство же ветвей кончается логическими тупиками. ПРОЛОГ решает задачу, обходя последовательно все ветви этого дерева. Свойство языка таково, что процесс

поиска решения по одной ветви практически не зависит от поиска по другой. Поэтому обход различных ветвей можно производить одновременно — параллельно! Если в процедурных языках стоит задача, как распараллелить выполнение по сути своей последовательной программы, то в языках логического программирования последовательно выполняется программа, которая по сути своей параллельна. На внутренне присущий языку ПРОЛОГ (и его потомкам) параллелизм возлагают большие надежды разработчики ЭВМ новых поколений. Прототипы параллельных Пролог-машин и существующие проекты обеспечивают повышение быстродействия еще в 5—10 раз по сравнению с последовательными Пролог-машинами.

Из сказанного должно быть ясно, почему японские ученые, предложившие в 1981 г. национальный проект по созданию машин нового поколения, ядром его выдвинули именно логическое программирование, которое, по их мнению, представляет собой новый универсальный принцип построения всей компьютерной науки. Как отмечают японские ученые Фучи и Фурукава [6], ставка на логическое программирование, сделанная в начале реализации проекта создания машин пятого поколения, оправдала себя. Более того, проект, который, правда, еще не завершен, обязан своим успешным прогрессом логическому программированию. Процесс выработки подходов к созданию машин новых поколений, говорят они, был похож на складывание картинки из детских кубиков. Когда в качестве основного подхода было выбрано логическое программирование, «кубики начали устанавливаться на свои места очень быстро».

Разумеется, развитие логического программирования еще не завершено, более того, оно только началось. Но тем не менее уже сейчас видна очевидная перспективность этого подхода и обеспечиваемые им преимущества.

Автор надеется, что эта статья дала читателю первое впечатление о логическом программировании. Теперь, подытоживая все сказанное, можно определить, что такое логическое программирование:

— это новый подход к программированию, обеспечивающий решение задач с помощью формализованной записи их условий;

— это унифицированный подход к построению баз данных и языков программирования;

— это аппарат построения систем искусственного интеллекта;

— это основы архитектуры сверхбыстродействующих ЭВМ будущего, снабженных базами знаний, вооруженных искусственным интеллектом, обеспечивающих с минимальными затратами решение сверхсложных задач, которые ставит перед человечеством сегодняшний день и еще поставит будущее.

Литература

1. ЭВМ пятого поколения. Концепции, проблемы, перспективы / Под ред. Т. Мото-ока. — М.: Финансы и статистика, 1984. — 110 с.

2. Клоксин У., Меллиш К. Программирование на языке Пролог. — М.: Мир, 1987. — 336 с.

3. Дейт К. Введение в системы баз данных. — М.: Наука, 1980. — 463 с.

4. Смаллиан Р. М. Алиса в стране смекалки. — М.: Мир, 1987. — 191 с.

5. Dobry T. P., Despain A. M., Patt Y. N. Performance studies of a Prolog machine architecture. — SIGARCH Newsletter, v. 13, n. 3, 1985, p. 180—190.

6. Fuchi K., Furukawa K. The role of logic programming in the fifth generation computer project. — New generation computing, n. 5, 1987, p. 3—28.





В последнее время наряду с компьютерами БК-0010, работающими на ФОКАЛе, получили распространение компьютеры БК-0010-01, в ПЗУ которых «зашит» язык программирования БЕЙСИК.

ВКЛЮЧЕНИЕ ПОДПРОГРАММ В МАШИННЫХ КОДАХ В ПРОГРАММЫ НА БЕЙСИКе ДЛЯ БК-0010-01

По сравнению с ФОКАЛом БЕЙСИК работает значительно быстрее, программисту представляются более широкие возможности, за которые, правда, приходится расплачиваться двукратным уменьшением объема памяти для программ. Тем не менее имеющейся памяти вполне достаточно для создания сложных вычислительных программ и логических игр, а популярность БЕЙСИКА столь велика, что при наличии выбора ему отдается явное предпочтение перед ФОКАЛом.

Многих владельцев БК привлекает имеющаяся в БЕЙСИКе возможность использования подпрограмм в машинных кодах, однако далеко не все знают, как к этому подступиться, тем более что по необъяснимому капризу разработчиков операторы BLOAD и BSAVE для чтения и записи на магнитную ленту двоичных файлов действуют только в директивном режиме, то есть включить их в программу нельзя. Кроме того, для выделения памяти подпрограммам в машинных кодах приходится устанавливать границы Бейсик-системы оператором CLEAR, что тоже возможно лишь в директивном режиме. При использовании «штатных» возможностей БЕЙСИКа, загрузка и запуск таких программ требуют множества предварительных манипуляций с клавиатурой и магнитофоном, что отнимает много времени, увеличивает вероятность ошибок и в конечном итоге приводит пользователей к решению обойтись без использования кодовых подпрограмм.

Между тем существуют сравнительно несложные способы преодоления указанных трудностей. Во-первых, если подпрограмма в кодах занимает в памяти не более 20—30 машинных слов, ее можно не записывать на магнитофон, а включать прямо в текст программы на БЕЙСИКе при помощи оператора Data и затем переслать в нужный участок ОЗУ, используя оператор непосредственного доступа к памяти POKE.

Во-вторых, достаточно длинные подпрограммы в кодах можно «спрятать» в стековой области БЕЙСИКа, которая занимает адреса с 400 до 2000 (здесь и далее адреса восьмеричные). Такой объем стека, судя по всему, связан с предполагаемым использованием данной версии БЕЙСИКа в новой модели БК-0011 с оперативной памятью до 128 кбайт. В самых длинных программах на БК-0010 обычно используется не более трети выделенной под стек памяти. Это означает, что есть возможность без особого риска располагать подпрограммы в кодах длиной до 500 байт в младших адресах стека, не затрагивая программную память, т. е. не используя оператор CLEAR.

Эти два приема помогут реализовать некоторые несуществующие в БЕЙСИКе операции. К примеру, часто владельцев БК огорчает недоступность служебной строки экрана, которую БЕЙСИК использует для вывода первых символов программируемых ключей. Короткая подпрограмма в машинных кодах позволит нам использовать служебную строку для вывода любой символьной информации. Вот ее текст в мнемонике АССЕМБЛЕРА и в машинных кодах.

АССЕМБЛЕР		Код	
		Восьмеричный	Десятичный
CLR	R1	5001	2561
MOV	(R5)+, R3	12503	5443
MOV	(R5), R2	11502	4930
MET: MOVB	(R2)+, R0	112200	−27520
EMT	22	104022	−30702
INC	R1	5201	2689
SOB	P3, MET	77304	32452
RTS	R7	207	135

Нам остается разместить эту подпрограмму в памяти, начиная с адреса 400, и определить ее при помощи оператора БЕЙСИКа DEF USR. Участок программы на БЕЙСИКе, реализующий вывод в служебную строку текста «Электроника-БК-0010», будет выглядеть так:

```
10 DATA 2561,5443,4930,-27520,-30702,2689,32452,135
20 FOR I%=0 TO 7
30 READ J%
40 POKE &0400+2*I%, J%
50 NEXT I%
60 DEF USRO=&0400
70 A¤ = "ЭЛЕКТРОНИКА БК-0010"
80 A¤=USRO(A¤)
```

Аргументом функции USRO может быть символьная константа, переменная или выражение. Не допускается только использование символьных функций.

Теперь нам ничто не мешает составить подпрограмму для автоматической загрузки с магнитной ленты более длинных подпрограмм, которые нерационально размещать в операторе DATA. Собственно говоря, ее даже не надо составлять, такая подпрограмма есть в ПЗУ монитора и имеет стартовый адрес 100602. К ней можно обращаться из БЕЙСИК-программы, предварительно определив: DEF USR1 = 80100602. Следует помнить, что в основе всех операций с магнитофоном в БК-0010 лежит макрокоманда монитора EMT 36, которая использует блок параметров драйвера магнитофона, располагаемый по умолчанию в ячейках с адресами 320 — 371. В младший байт ячейки 320 записывается число 3 — код команды чтения файла с магнитной ленты. В старший байт монитор помещает результат операции: 0 — ошибок нет, 2 — при чтении не совпала контрольная сумма, 4 — операция прервана нажатием клавиши «СТОП». Далее, в ячейке 322 должен находиться начальный адрес памяти, куда считывается файл. Область с адресами 326—344 предназначена для хранения имени считываемого файла. Перед обращением к подпрограмме нам необходимо поместить в блок параметров все необходимые значения. Код команды и адрес загрузки отправим по нужным адресам оператором POKE. Имя загружаемого файла также можно переслать на место операторами БЕЙСИКа, однако программа пересылки получается громоздкой, да и соответствующая функция БЕЙСИКа MID ¤ ведет себя внутри цикла порой совершенно непредсказуемо. Для этой цели удобнее определить еще одну подпрограмму в машинных кодах, которая почти не отличается от предыдущей (см. таблицу).

В качестве примера рассмотрим программу на БЕЙСИКе, которая после старта автоматически переходит в режим загрузки с магнитофона подпрограммы в машинных кодах с именем FILENAME, после успешной загрузки выполняет подпрограмму, а в случае ошибки при чтении загружает ее повторно.

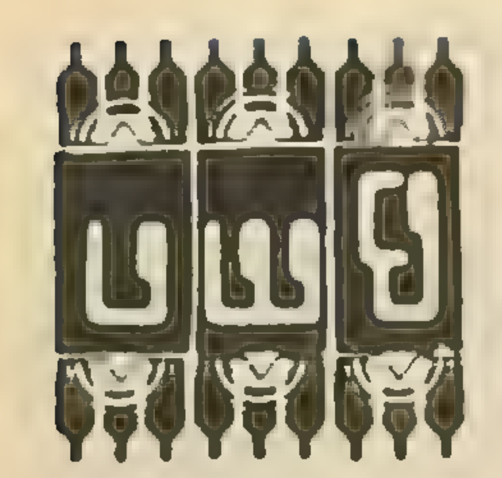
```
10 DATA 5443,4930,5569,214,-27503,32450,135
20 FOR I%=0 TO 6
30 READ J%
40 POKE &0400+2 * I%, J%
50 NEXT I%
60 DEF USR1=&0400
70 DEF USR2=&0100602
80 POKE &0320,3
90 POKE &0322,&0420
100 A¤ = USR1("FILENAME
110 I%=USR2(I%)
120 IF PEEK(&0320) \ 256 > O THEM 110
130 DEF USR3=&0420
140 I%=USR3(I%)
```

Имя файла в качестве параметра в строке 100 рекомендуется дополнять справа пробелами до 16 знаков.

Стоит указать еще одну область ОЗУ, куда можно загружать подпрограммы в кодах длиной до 256 байт, не изменяя границу БЕЙСИК-системы оператором CLEAR. Это так называемый буфер ввода-вывода, расположенный в ячейках с адресами 37400—37776. Здесь, однако, есть два ограничения: в программе не должно быть операторов чтения и записи данных на магнитную ленту INPUT# и PRINT#, а саму программу нельзя записывать на ленту директивой SAVE, поскольку буфер ввода-вывода предназначен именно для этих команд. К слову говоря, записывать БЕЙСИК-программы на ленту во всех отношениях удобней директивой CSAVE: экономятся время, лента и меньше вероятность ошибок.

АССЕМБЛЕР		Код	
		Восьмеричный	Десятичный
MOV	(R5)+, R3	12503	5443
MOV	(R5), R2	11502	4930
MOV	≠326, R1	12701	5569
		326	214
MET: MOVB	R2+, (R1)+	112221	−27503
SOB	R3, MET	77302	32450
RTS	R7	207	135

Д. БАРОНОВ,
член Московского клуба
пользователей бытовых компьютеров



Бытовая ЭВМ «Электроника БК-0010» поставляется пользователям оснащенной интерпретатором одного из языков высокого уровня — ФОКАЛ или БЕЙСИК. Эти языки служат очень хорошим инструментом для разработки различного рода программного обеспечения — игр, научно-технических расчетов, демонстрационных и обучающих программ и т. п. Однако существует область приложения, в которой эти языки использовать весьма сложно и неэффективно: создание системных программ (копировщиков файлов, новых интерпретирующих и компилирующих систем), программ с динамической графикой, текстовых редакторов, машинно-независимых программ (работающих как на БК-0010, так и на БК-0010-01). Поэтому практически одновременно с появлением БК-0010 возникла острая необходимость в средствах, покрывающих этот пробел.

АССЕМБЛЕР ДЛЯ ЭВМ «ЭЛЕКТРОНИКА БК-0010»

Встроенные средства программирования низкого уровня. Некоторые, хотя и очень слабые, возможности программирования БК-0010 на уровне машинных команд предоставлял зашитый в ПЗУ отладочный монитор. Он позволял записывать и читать в ячейках памяти ЭВМ машинные команды и данные и запускать введенную программу. Этого было явно недостаточно. Работать с таким средством могли только очень опытные пользователи, хорошо владеющие программированием в машинных кодах. Вводимая и впоследствии распечатываемая программа представляла собой ряд восьмеричных цифр, что во много раз снижало эффективность труда программиста. Организация ветвлений представляла одну из самых больших трудностей при разработке программы. В довершение всего практически полностью отсутствовали средства обнаружения ошибок. Все эти проблемы привели к тому, что появилось средство, существенно облегчавшее разработку программ в машинных кодах, — семейство программ типа ОТЛАДЧИК.

Язык АССЕМБЛЕРа и его интерпретаторы. Одно из главных достоинств программы типа отладчик в том, что она перестала представлять собой последовательность малопонятных восьмеричных чисел. Для каждой машинной команды появился свой символьный код, для каждого типа операндов команды — свое условное обозначение. Появился первый язык АССЕМБЛЕРа для БК-0010.

Что такое язык АССЕМБЛЕРа и как он соотносится с другими языками программирования? Когда мы пишем программу на языке высокого уровня, мы не задумываемся о «мелочах» — работаем ли мы с целыми числами или с вещественными, готова ли ЭВМ вывести очередной символ на экран или нет. Мы просто пишем «+» или «−» или оператор WRITE. Строки исходного текста программы попадают затем на обработку специальной программе — компилятору. Компилятор вместо одной нашей исходной строчки порождает целую программу. Когда потребуется, эта программа преобразует к одинаковому виду различные числа, соединенные операцией «+». К тому же она сама будет проверять, можно ли выводить очередной символ на экран. В итоге из каждой строчки программы на языке высокого уровня получаются многие машинные команды (порождаются десятки, а то и сотни машинных команд).

После обработки строчки программы на языке АССЕМБЛЕРа специальной программой-ассемблером получается тоже машинная команда, но только одна! Каждая строчка программы языка АССЕМБЛЕРа превращается в одну машинную команду. Конечно, написать программу вычисления синусов на языке АССЕМБЛЕРа для БК-0010 очень сложно — ведь эта ЭВМ на уровне машинных команд умеет только складывать и вычитать. А вот для подсчета контрольной суммы файла, например, АССЕМБЛЕР очень удобен.

Программа-отладчик принимает исходный текст на языке АССЕМБЛЕРа, производит синтаксический контроль команд, перекодирует вводимую информацию из символьного представления в машинные коды и располагает эти коды в оперативной памяти. Программист, разрабатывающий программу, видит на экране телевизора текст программы и адреса ячеек памяти, в которые помещается перекодированная программа. Когда ввод программы завершен, можно заставить ее выполняться.

Отладчик располагает самыми различными режимами для выполнения программ, что и облегчает ее отладку. Программу можно просто выполнить без всяких хитростей. Если что-либо в программе не устраивает программиста, он может заставить программу выполняться с различной скоростью, в том числе и очень медленно, чтобы успеть рассмотреть все, что выдается на экран во время выполнения программы. Наконец, программу можно заставить выполняться по шагам, с ожиданием после выполнения каждой команды специального разрешения программиста на выполнение следующей команды. Если возникла необходимость, можно проверить содержимое регистров процессора и любой ячейки памяти, а также распечатать содержимое памяти на принтере. Такие услуги делают отладчик очень ценным помощником при написании программ и их проверке. Когда «последняя» ошибка найдена, отладчик предоставляет возможность записать окончательный вариант программы на магнитофон.

Сегодня известно более десятка программ семейства ОТЛАДЧИК. Принцип действия всех их одинаков, они различаются предоставляемым набором услуг и количеством невыявленных ошибок. Как правило, чем больше номер, указанный в названии программы, тем эти показатели лучше.

Последняя известная автору программа этой серии — ОТЛАДЧИК12.

АССЕМБЛЕР МИКРО10. Другой подход к созданию программ в машинных кодах воплощают АССЕМБЛЕРы семейства МИКРО. Этот более традиционный подход заключается в том, что исходный текст программы на языке АССЕМБЛЕРа просто преобразуется в машинные команды программой-ассемблером, а затем получившаяся последовательность машинных команд загружается в память ЭВМ и самостоятельно выполняется, без участия какого-либо контроля со стороны АССЕМБЛЕРа. В таком варианте у нас отсутствует возможность осуществлять замедленный или пошаговый прогон программы, невозможно также в ходе отладки изменить текст программы. Какие же преимущества дает такой подход?

Во-первых, существуют некоторые типы программ, выполнять которые под управлением отладчика очень сложно. Это, например, программы, работающие с использованием механизма прерываний.

Во-вторых, отладчик позволяет создавать и выполнять программы лишь такой длины, которые помещаются в оставшееся от самого отладчика место в оперативной памяти машины. А ведь такая сложная программа, как отладчик, занимает довольно много места — около 4—5 кбайт.

Эти существенные трудности позволяет преодолеть программа-ассемблер. Правда, преодолевает их она не совсем обычным способом: просто во время работы программы пользователя самого АССЕМБЛЕРа в памяти машины нет, потому-то ничто не мешает занимать программе всю память, вызывать прерывания и вообще делать все, что угодно.

Еще один положительный эффект от применения **АССЕМБЛЕРа** состоит в том, что пользователь может применять для написания программ очень богатый язык. В язык АССЕМБЛЕРа в таком случае входят не только коды для обозначений машинных команд и их операндов, но также метки, которые позволяют обозначить ячейки памяти машины именами для более удобного их использования; специальные директивы, организующие размещение в памяти текстов, чисел, таблиц; комментарии, делающие программу несравненно более понятной окружающим и даже самому автору; многие другие полезные вещи, превращающие написание программы в достаточно приятный процесс.

Известен целый ряд АССЕМБЛЕРов семейства МИКРО, разработанного С. В. Шмытовым и А. Н. Сомовым (Москва). Рассмотрим подробнее одну из последних программ — АССЕМБЛЕР МИКРО10.

МИКРО10 представляет собой законченную систему для создания программ в машинных кодах. Эта система содержит три основные компоненты: редактор текста, ассемблер (компилятор языка АССЕМБЛЕР) и редактор связей. В принципе это три совершенно самостоятельные программы, которые в условиях БК-0010 оказалось выгодным собрать воедино, чтобы исключить по возможности работу с магнитофоном при переходе от одной стадии разработки программы к другой.

Обычно сеанс работы с МИКРО10 начинается с того, что с помощью специальной команды пользователь запускает текстовый редактор. Эта программа служит для ввода исходного текста программы в ЭВМ. Текстовый редактор МИКРО10 построен на базе широкоизвестного текстового редактора EDASP. Поэтому исходные тексты программы доступны также для работы в редакторе EDASP. После ввода исходного текста пользователь обычно осуществляет специальной командой его компиляцию, или ассемблирование. При этом запускается другая программа системы — компилятор (ассемблер). Компилятор просматривает последовательно строку за строкой, преобразуя символьные коды пользователя в машинные команды. При этом он создает специальную таблицу, куда записывает все ячейки памяти, которым пользователь присвоил имена, а также имена из других программ. С одной стороны, наличие таких имен может говорить об ошибке, а с другой, возможно, вы хотели обратиться к ячейке, расположенной в программе, которую написали вчера (или только собираетесь написать). Имена ячеек, находящихся в других программах, называют неразрешенными ссылками. Для того чтобы указать действительный адрес неразрешенной ссылки, используется третья компонента системы — редактор связей.

Редактор связей обычно применяют тогда, когда исходный текст программы не помещается целиком в память. Исходный текст занимает намного больше места, чем получаемая из него машинная программа. Поэтому большую программу пишут небольшими порциями и по отдельности компилируют. Вместе с командами компилятор записывает таблицу имен с соответствующими именам адресами. Таким образом мы получаем много небольших кусков одной большой программы, причем каждый кусок содержит таблицу имен, которыми он располагает, и имен, адреса которых он пока не имеет. Редактор связей производит компоновку большой программы. При этом требуемые адреса в одних программах он делает известными, просматривая таблицы имен в остальных программах. В итоге все требуемые адреса будут где-то найдены, или, как говорят, все ссылки должны быть разрешены. Программа становится единой и работоспособной. Ее можно запустить тотчас же или записать до лучших времен на магнитофон.

Система МИКРО10 занимает в памяти около 4,5 кбайта и позволяет создавать программы, занимающие до 16,5 кбайта, т. е. всю доступную в нормальном режиме память. Если объем создаваемой программы не слишком велик, всю разработку можно произвести без записи на магнитофон промежуточных результатов. То есть после ввода программы, компиляции и пробного запуска допускается снова войти в систему с так называемого адреса «теплового пуска», исправить редактором текста найденные ошибки и снова повторить компиляцию, запуск и т. д.

Нужен ли вам язык АССЕМБЛЕРа? Язык АССЕМБЛЕРа, бесспорно является очень мощным и сложным средством, поэтому его применение не случайно называют «высшим пилотажем программирования». Но как и все в нашем мире, он имеет свои слабые стороны и ограничения.

Во-первых, неверно широко распространенное мнение о том, что программы, написанные на языке АССЕМБЛЕРа, короче и быстрее программ,

написанных на языках высокого уровня. Интерпретирующие системы дадут сто очков вперед АССЕМБЛЕРам по компактности исходного текста. Напишите на БЕЙСИКе команду PRINT SIN (.33) — эта коротенькая программа в исходном виде занимает у вас всего несколько байт. Попробуйте же написать эквивалентную программу на языке АССЕМБЛЕРа — трудно сказать, сколько сотен строк она будет иметь. Кроме того, удачно написанные трансляторы языков высокого уровня создают программы, улучшить которые практически невозможно, так что человек, средне владеющий языком АССЕМБЛЕРа, создаст программу наверняка менее эффективную, чем транслятор.

Во-вторых, писать на языке АССЕМБЛЕРа намного дольше, чем на языках высокого уровня — ведь пользователю приходится собирать более или менее осмысленные действия из самых элементарных машинных команд.

Наконец, ввиду вышесказанного язык АССЕМБЛЕРа как ни один другой язык с готовностью предоставляет вам возможность сделать самые разнообразные ошибки, выловить которые довольно сложно.

Вот почему не следует думать, что АССЕМБЛЕР — панацея и, познакомившись с ним, вы будете уметь все. Немногие могут сейчас доказать теорему Пифагора (не столь уж и сложную) — а ведь аксиомы геометрии помнят почти все.

Для чего же нужен язык АССЕМБЛЕРа? Прежде всего он позволяет создавать программы, работающие быстрее, чем написанные на любом интерпретируемом языке — ФОКАЛе, БЕЙСИКе, ФОРТе и пр.

Во-вторых, для создания программ, требующих хитроумных приемов (для повышения эффективности).

В-третьих, для написания самоизменяющихся программ, что отчасти позволяет преодолеть ограниченность набора команд.

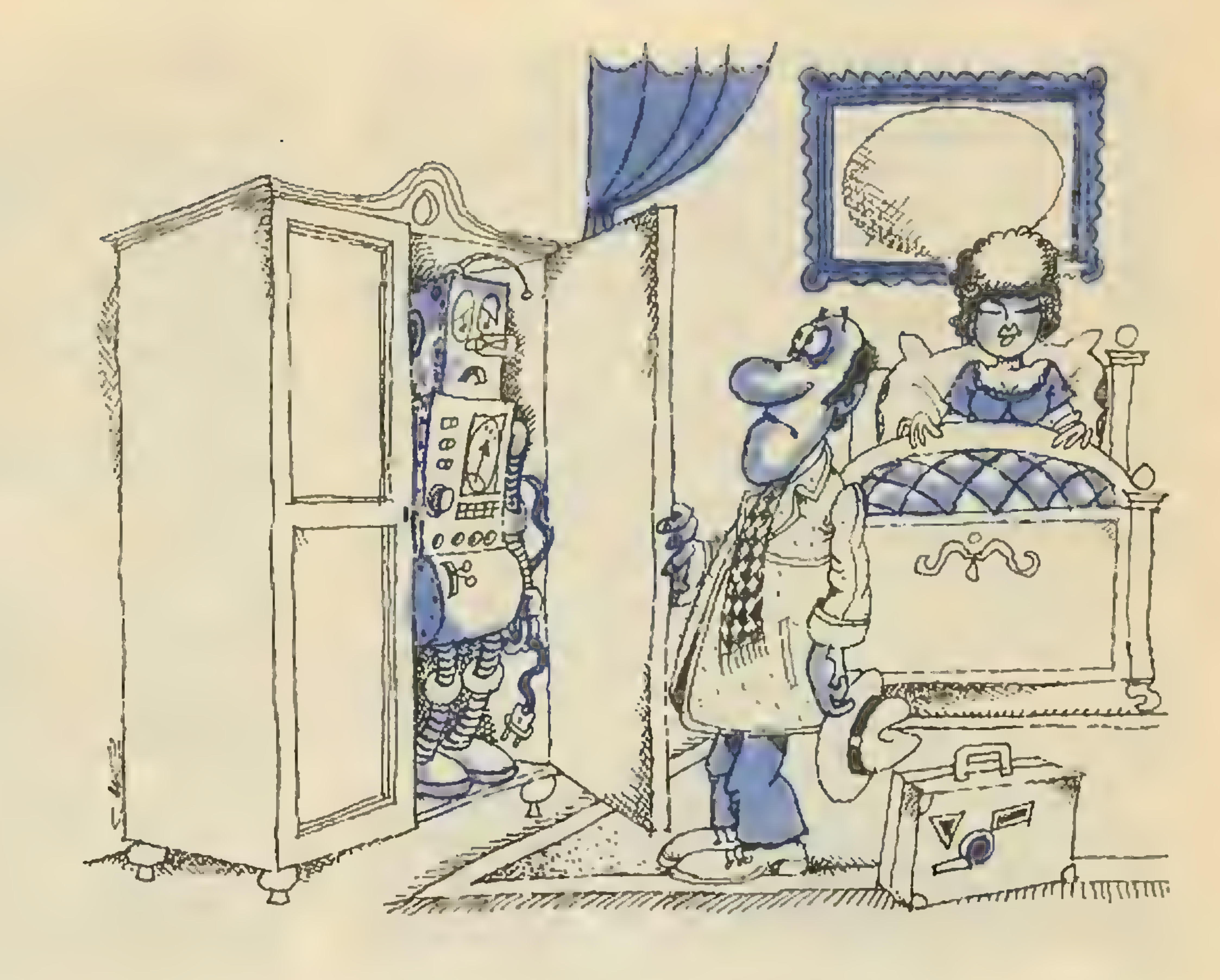
В-четвертых, для программ, напрямую работающих с каким-то внешним устройством (электронным диском, дополнительной клавиатурой и т. п.).

Наконец, язык АССЕМБЛЕРа нужен тем, кто хочет до конца понять устройство и принципы работы своей ЭВМ, ибо в этом деле профессионализм особенно необходим, даже если он проявляется в свободное от работы время.

В. В. СЛОБОДЧУК

Уважаемые читатели! В выпуске № 4 нашей серии в рубрике «Как убить машинное время» ошибочно помещена программа, не имеющая отношения к тексту описания игры «Кольцо с секретом». Программа, соответствующая этому тексту, будет опубликована в выпуске № 11 серии «Вычислительная техника и ее применение». В № 6 серии «ЭВМ в ускорении научно-технического прогресса» неправильно даны сведения об одном из авторов статьи «Информационная индустрия в научно-техническом прогрессе общества» Г. В. Красовском. Должно быть: Красовский Геннадий Владимирович, кандидат экономических наук, работает в НИИ планирования и нормативов при Госплане СССР. Редакция приносит извинения за допущенные ошибки.

«КОМПЬЮТЕРНЫЕ ВИРУСЫ»



На мир компьютеров обрушилась великая напасть, способная уничтожать записанные программы и блоки памяти. Самое поразительное в том, что болезнь может передаваться от одного компьютера другому через дискетты, которые становятся разносчиками «компьютерного вируса».

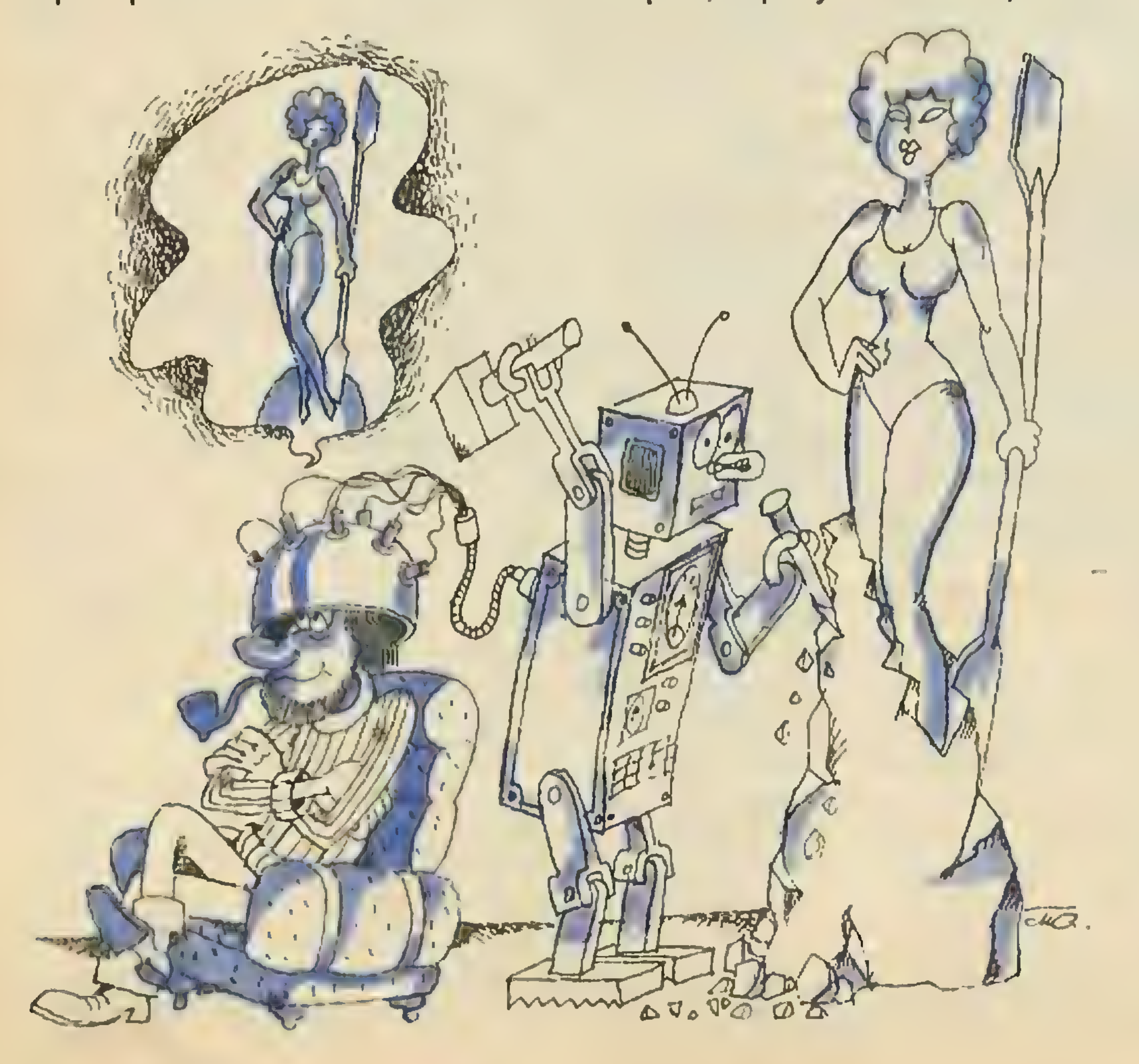
Более того, вирус может подвергаться мутации, и тогда просто невозможно проследить его истоки, выяснить, с какой дискетты он пришел.

Создатель вируса Фред Коэн, компьютерщик из университета в Цинциннати, штат Огайо. Он изобрел вирус лет пять назад, сделал это в порядке эксперимента, не подозревая даже, к каким последствиям может привести его изобретение. «Вирус» — это специальная, записываемая на дискетты программа, способная интегрировать сама себя в другие программы, которые она обнаружит в чреве компьютера. Интегрировать и уничтожать их.

«Здоровая» дискетта, вставленная в дисковод компьютера, где побывал вирус, заражается. Болезнь идет дальше и дальше. Ей безразлично, что уничтожать: это могут быть программы бронирования билетов в компьютерных системах авиакомпаний, это могут быть блоки памяти, которыми пользуется банк, это могут быть занесенные на дискетты карты болезней пациентов клиник. Это может быть все, что угодно.

По свидетельству Эй-би-си, «один из госпиталей на восточном побережье Соединенных Штатов потерял уже 40 процентов объема своей картотеки... В целом же, как подсчитали эксперты, в результате вторжения компьютерного вируса различные компании, предприятия, учреждения Америки понесли убытки на сумму в 20 миллионов долларов».

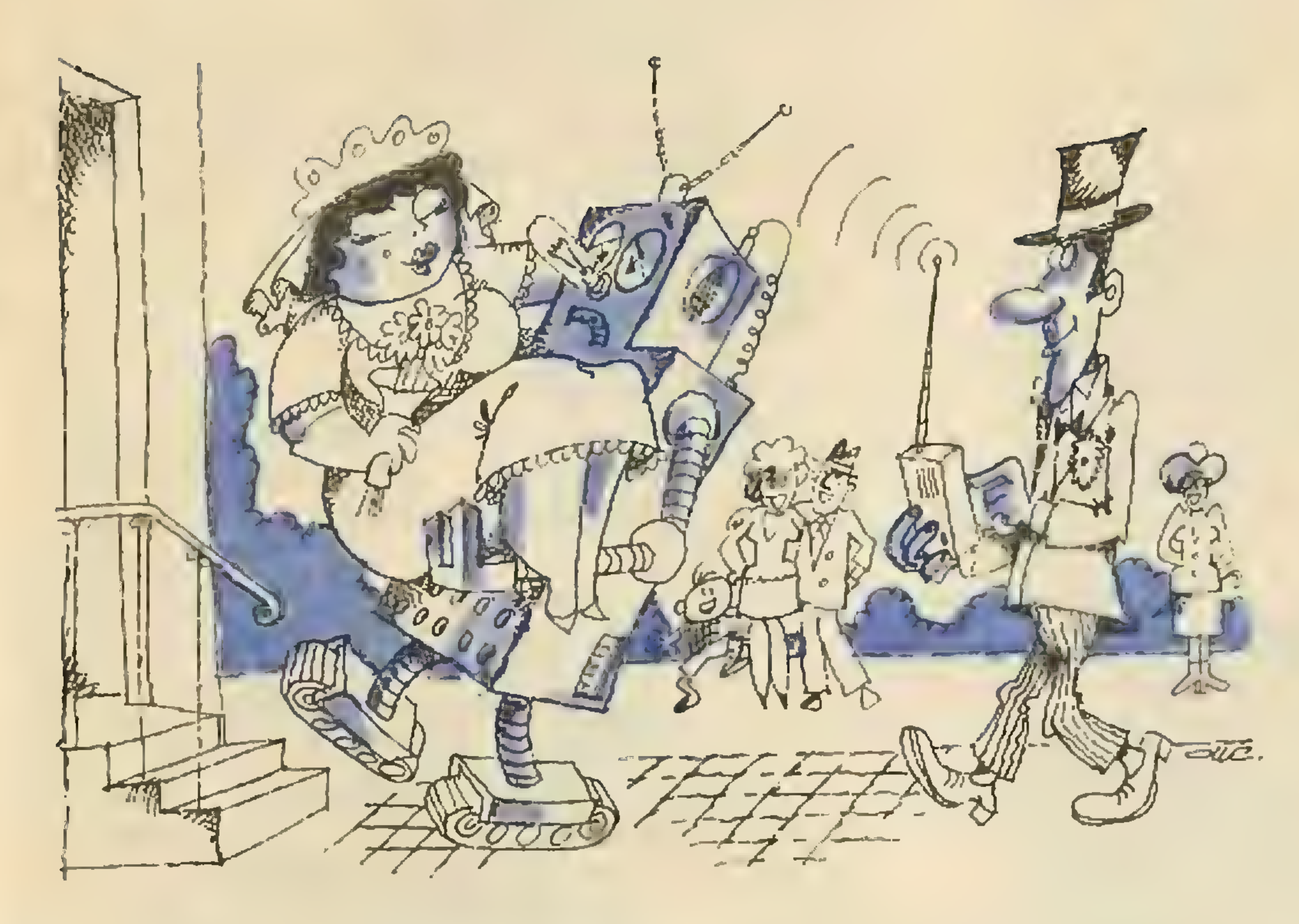
«Но, — заключает свой репортаж корреспондент телекомпании, — убытки будут расти: поскольку компьютерный вирус распространяется так же, как и вирус настоящий, избавиться от него будет столь же трудно».



И еще одна цифра. Ее привел «отец» вируса Фред Коэн: без особой гордости он признал, что эта напасть может в течение нескольких недель уничтожить миллионы компьютерных программ.

Так, в конце 1987 г., как считают, из политических соображений ложная информация «заразила» несколько ЭВМ в Древнееврейском университете (Иерусалим, Израиль). Этот факт был обнаружен, когда машинные программы начали по непонятным причинам переполнять свои магнитные диски, в результате чего существенно увеличился объем программных файлов. Несмотря на разработку и внедрение защитных машинных программ, ожидают, что введенная паразитная информация вызовет массовое уничтожение ценной информации.

В Лехайском университете (штат Пенсильвания) паразитная информация вызвала уничтожение сотен программ на магнитных дисках и стирание



множества файлов, подготовленных студентами и преподавательским составом факультета вычислительной техники. «Вирус» был обнаружен в ноябре 1987 г., когда в библиотеку было возвращено большое количество магнитных дисков с ошибочными программами. Для уничтожения паразитной информации разработана специальная машинная программа, но специалисты не дают гарантий от повторения подобной ситуации, поскольку доступ к вычислительной технике в лабораториях университета имеет большое число студентов.

В Пенсильванском университете (США) также было замечено несколько случаев введения ложной информации, которая обусловила изменения наименований файлов или заполнение больших магнитных дисков во многих персональных ЭВМ. Аналогичная паразитная информация в виде новогоднего поздравления обусловила перегрузку вычислительных машин в электронной почте фирмы «Интернешнл бизнес мэшинз» в штате Флорида (США).

Для защиты от ложной информации владельцам ЭВМ, кроме пользования продукцией респектабельных поставщиков машинных программ, специалисты фирмы «SRI интернешнл» рекомендуют применять специальные тестовые программы для проверки на отсутствие изменений параметров с момента исполнения в последний раз рабочей программы или исполнение новых программ на отдельной изолированной вычислительной машине.

Для этих же целей служит разработанное фирмой «Лазертрив» (штат Нью-Джерси) специальное устройство, которое производит проверку рабочих программ перед их исполнением в ЭВМ и которое включает аварийную сигнализацию в случае обнаружения ложной информации.

Фирма «Спектра-секьюр трансмишн» (Швеция) разработала специальный комплект машинных программ, обеспечивающий как защиту вычислительных систем от ложной информации, так и восстановление «пораженных» машинных программ.

Этот комплект способен контролировать прохождение данных в информационной системе и выявить ошибки в электронных ключах каждого информационного файла. А вводиться он может либо в качестве штатного средства для проверки, либо для оперативной сигнализации при выявлении поврежденных файлов данных. Комплект работает по принципу произвольного выбора ряда файлов для постоянной проверки с выполнением контроля остальных файлов время от времени.







ПОДОБНЫ ЛИ ТРЕУГОЛЬНИКИ? ОТВЕТ ДАЕТ ЭВМ И ПОНОМАРЕНКО ЮРА (г. Одесса)

Если треугольники расположены на плоскости так, как показано на рис. 3, то их подобие очевидно. Труднее выяснить подобие треугольников, если они расположены на плоскости произвольным образом, например так, как показано на рис. 4.

Ниже предлагается программа, работая по которой ЭВМ выяснит, является ли треугольник ABC подобным треугольнику DEF. Треугольники задаются координатами своих вершин.

Сначала ЭВМ предполагает, что для стороны AB сходственной является сторона DE и составляет отношение R1=AB / DE. Затем ЭВМ предполагает, что для стороны BC сходственная сторона FE, вычисляет отношение R2=BC / FE, и если R1=R2, то переходит к вычислению R3=АЦ / DF и проверке условия R1=R3. Может оказаться, что R1=R2=R3 и тогда ЭВМ выдает сообщение: «Треугольники подобны, коэффициент подобия K=R1».

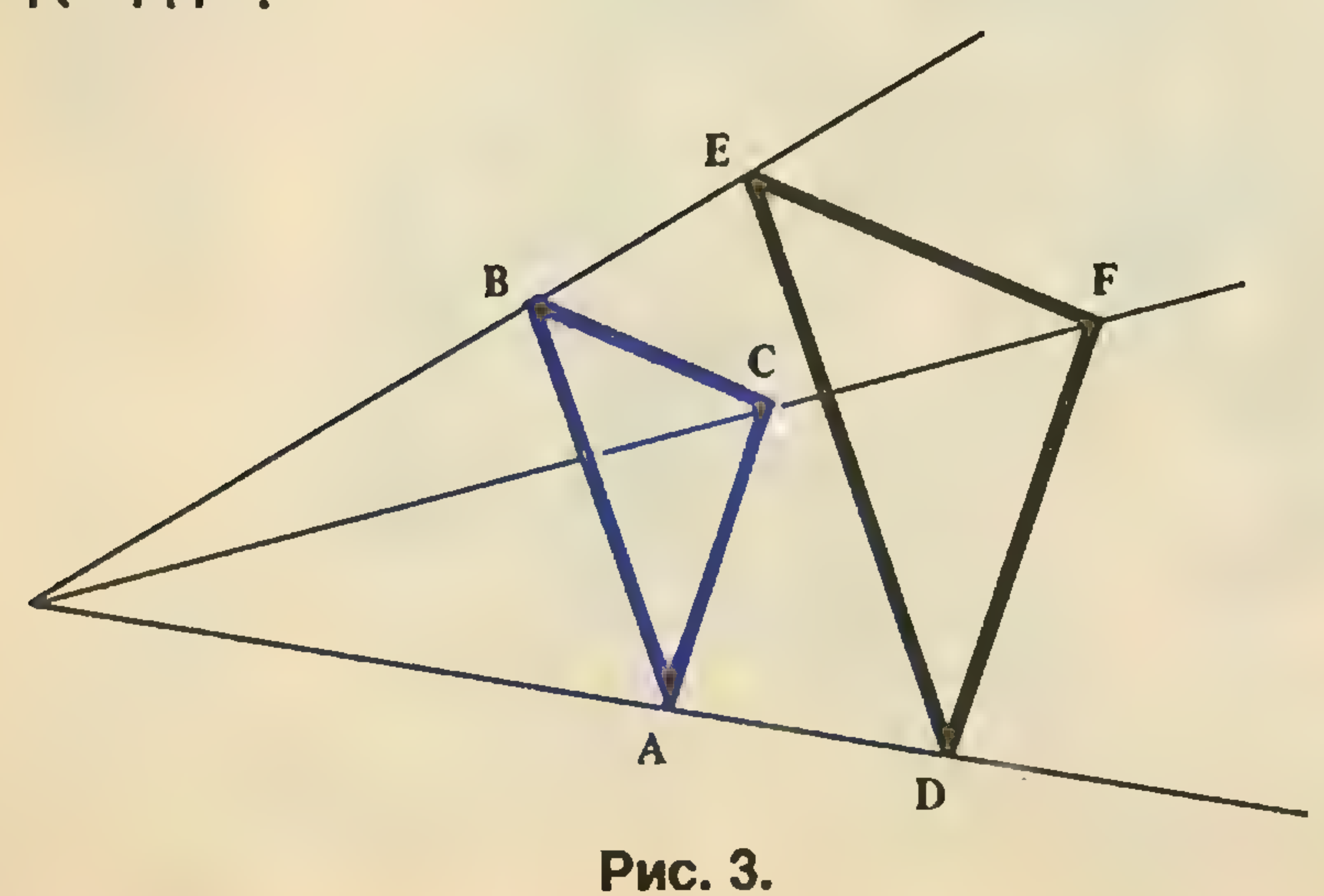


Рис. 3.

Рис. 4.

Если предположение, что стороны AB и DE сходственны, к такому ответу не привело, то ЭВМ предполагает, что сторона AB сходственна со стороной DF и проводит такой же анализ, как и ранее. Если и это предположение не приводит к выводу, что треугольники подобны, то в ход идет последнее предположение: для стороны AB сходственной будет сторона EF. После проверки этого предположения делается окончательный вывод о подобии треугольников.

Завершая работу, ЭВМ выводит изображение треугольников на ЭКРАН.

Текст программы:

```
10 REM ВЫЯСНЕНИЕ ПОДОБИЯ ТРЕУГОЛЬНИКОВ
20 SCREEN 0:COLOR 10,8
30 REM ВВОД КООРДИНАТ ВЕРШИН ТРЕУГОЛЬНИКОВ
40 PRINT "ВВОД КООРДИНАТ ВЕРШИН ТРЕУГОЛЬНИКОВ"
50 INPUT "XA, YA – ";XA, YA
60 INPUT "XB, YB – ";XB, YB
70 INPUT "XC, YC – ";XC, YC
80 INPUT "XD, YD – ";XD, YD
90 INPUT "XE, YE – ";XE, YE
100 INPUT "XF, YF – ";XF, YF
```

```
110 REM НАХОЖДЕНИЕ ДЛИН СТОРОН
120 LET AB=SQR((XA-XB)*(XA-XB)+(YA-YB)*(YA-YB))
130 LET AC=SQR((XA-XC)*(XA-XC)+(YA-YC)*(YA-YC))
140 LET BC=SQR((XB-XC)*(XB-XC)+(YB-YC)*(YB-YC))
150 LET DE=SQR((XD-XE)*(XD-XE)+(YD-YE)*(YD-YE))
160 LET DF=SQR((XD-XF)*(XD-XF)+(YD-YF)*(YD-YF))
170 LET EF=SQR((XE-XF)*(XE-XF)+(YE-YF)*(YE-YE))
180 R1=AB/DE:R2=AB/EF:R3=AB/DF
190 IF BC/EF=R1 THEN IF AC/DF=R1 THEN 270
200 IF BC/DF=R1 THEN IF AC/EF=R1 THEN 270
210 IF BC/DF=R2 THEN IF AC/DE=R2 THEN 270
220 IF BC/DE=R2 THEN IF AC/DF=R2 THEN 270
230 IF BC/DE=R3 THEN IF AC/EF=R3 THEN 270
240 IF BC/EF=R3 THEN IF AC/DE=R3 THEN 270
250 PRINT "ТРЕУГОЛЬНИКИ НЕ ПОДОБНЫ"
260 GOTO 320
270 IF (R1=1 OR R2=1 OR R3=1) THEN 280 ELSE 300
280 PRINT "ТРЕУГОЛЬНИКИ РАВНЫ"
290 GOTO 320
300 PRINT "ТРЕУГОЛЬНИКИ ПОДОБНЫ"
310 PRINT "КОЭФФИЦИЕНТ ПОДОБИЯ К=": R1
320 STOP
330 SCREEN 2:COLOR 16,2
340 LINE (30,30)-(630,30),4
350 LINE (30,30)-(30,390),4
360 LINE (30+XA,30+YA)-(30+XB,30+YB),11
370 LINE (30+XB,30+YB)-(30+XC,30+YC),11
380 LINE (30+XA,30+YA)-(30+XC,30+YC),11
390 LINE (30+XD,30+YD)-(30+XE,30+YE),2
400 LINE (30+XE,30+YE)-(30+XF,30+YF),2
410 LINE (30+XF,30+YF)-(30+XD,30+YD),2
420 END
```

Комментарии специалиста (инж. А. С. Металиди). Школьник реализовал один из многих возможных алгоритмов для выяснения подобия треугольников. Можно было бы, например, сначала упорядочить стороны каждого треугольника по их длинам, затем найти отношение сходственных сторон. Имеются и другие подходы.

В качестве обобщения задачи предлагаем читателю самостоятельно внести изменения в текст программ, с тем чтобы ЭВМ смогла выяснить подобие треугольников, расположенных в пространстве.

Начинающим

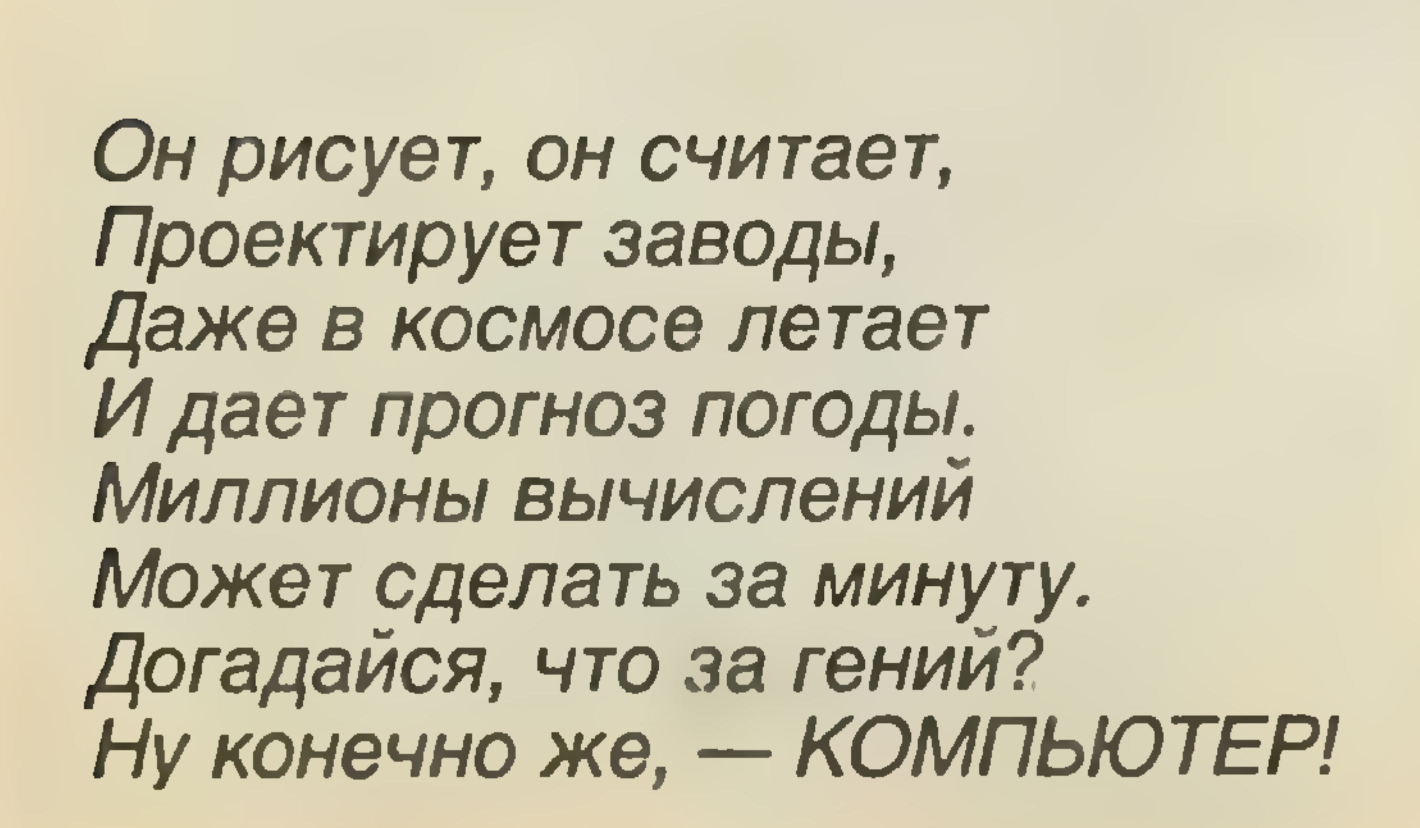
КОМПЬЮТЕР — электронная вычислительная машина. Он состоит из процессора, который перерабатывает поступающую информацию, памяти, где хранятся данные и программы их обработки, и разнообразных внешних устройств для связи с человеком, с другими компьютерами, с умными станками и машинами...

Весь процесс переработки информации в компьютере определяется программой, которую создает для него человек. В зависимости от этой программы компьютер становится помощником врача, конструктора, руководителя завода... — т. е. самым распространенным орудием труда человека!

Еще 40 лет назад, на заре своей истории компьютер был огромной «счетной фабрикой», занимавшей целый дом! Сейчас компьютер такой же производительности можно разместить в портфеле.

Многие думают, что компьютер предназначен в основном для расчетов. Ведь слово «компьюто» означает считать, вычислять. Однако главный его талант — работать со словами, строить фразы, редактировать тексты и даже... рисовать движущиеся картины на экране телевизора.





Он рисует, он считает,
Проектирует заводы,
Даже в космосе летает
И дает прогноз погоды.
Миллионы вычислений
Может сделать за минуту.
Догадайся, что за гений?
Ну конечно же, — КОМПЬЮТЕР!

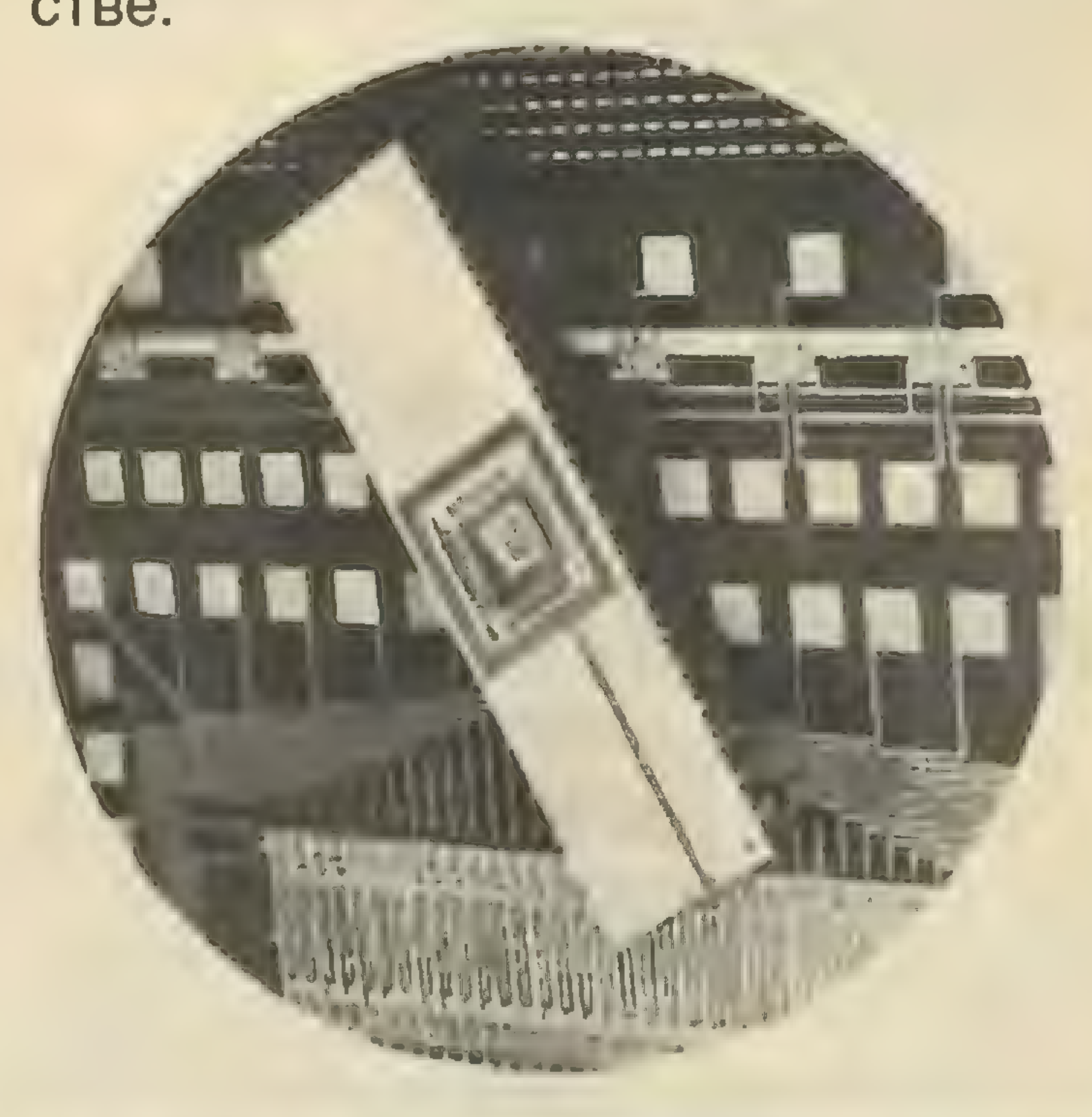
Вся информация хранится и перерабатывается в компьютере в виде электрических импульсов. Для этого работают в нем сотни и тысячи электронных схем. Миллион вычислительных операций в секунду может выполнить такая схема. А размеры ее не больше копеечной монеты.

ПРОЦЕССОР — главная часть компьютера, реализующая процесс переработки информации. Поступающую с терминалов информацию он преобразует в ряды чисел, которые посылает в память на хранение. Он выполняет операции, заданные программой, управляет всем вычислительным процессом и координирует работу всех других устройств. Процессор — «ум и сердце» компьютера.

Если вводить в процессор данные, набирая их на миниатюрных клавишах, а результат показывать на табло из маленьких цифр, получится обычный карманный калькулятор.

Если добавить процессору емкую память и общаться с ним с помощью дисплея, получится персональный компьютер.

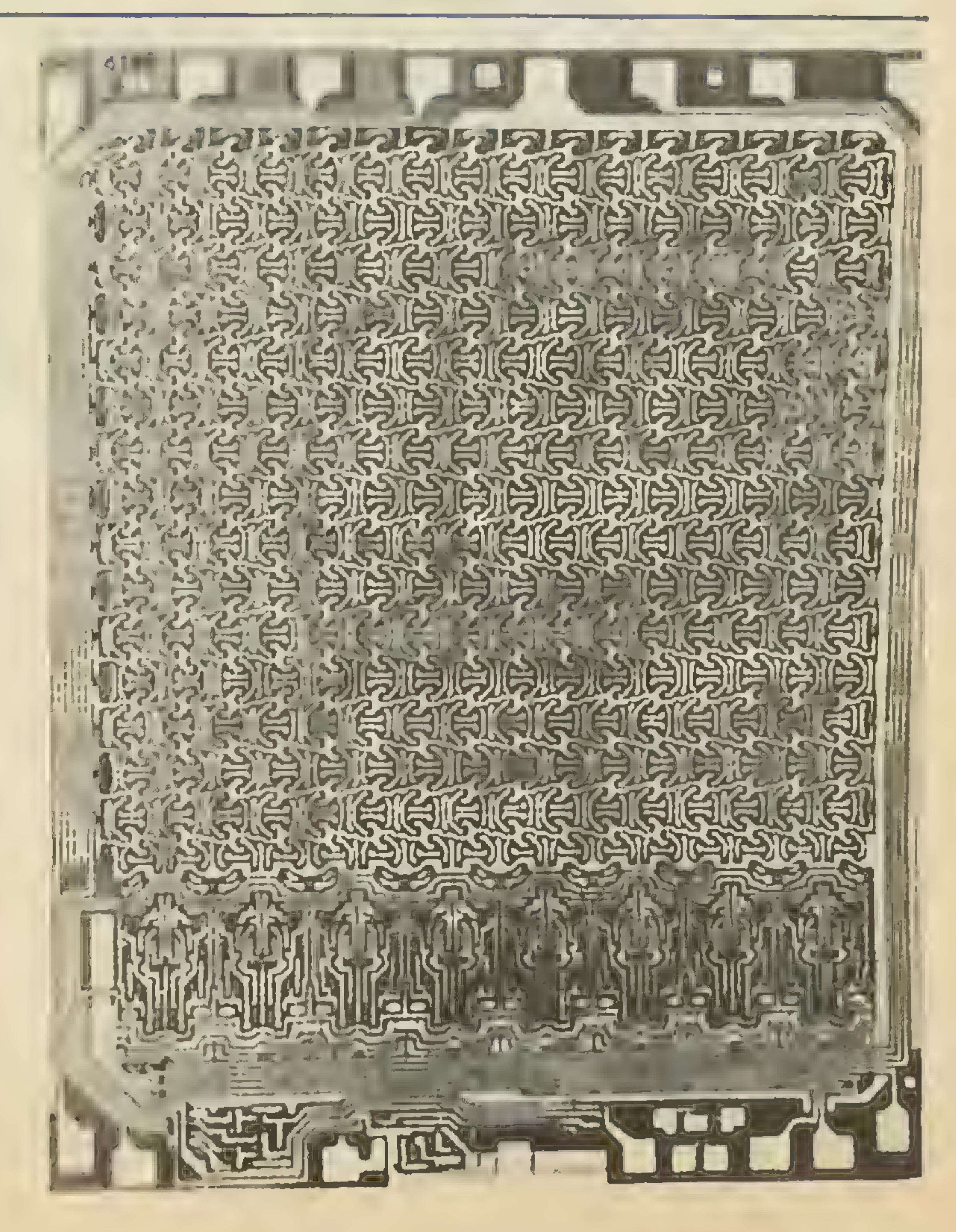
В современных суперкомпьютерах работает много процессоров. Они ведут решение задачи параллельно, передавая между собой промежуточные результаты, как детали на производстве.



Ум компьютера — процессор.
Как заправский эрудит,
Вычислительным процессом
Он легко руководит.
Чтобы выполнить программу,
Он считал, запоминал
И ответ, как телеграмму,
Передал на терминал.

ПАМЯТЬ — так называют запоминающее устройство компьютера. Память — своеобразные электронные соты, куда электрические импульсы несут информацию. Внутренняя память — как записная книжка компьютера, которая всегда под рукой. Состоит она из тысяч электронных ячеек, каждая со своим номером, как квартиры в большом доме. Здесь хранится выполня-

Здесь уютно, как в каюте,
Информация живет.
Чудо-импульсы компьютер
Сам сюда передает.
И они гурьбой веселой
Возникают там и тут,
В память импульсы, как пчелы,
Информацию несут.
Сохранится в таких сотах
Все на долгие года.
Человек забудет что-то,
А компьютер — никогда!



емая программа и данные для ее работы. Внешняя память компьютера — целая библиотека с тысячью томов, информация там хранится про запас, на всякий случай. Для внешней памяти используются магнитофон с кассетами или дисковод с маленькими магнитными дисками, вроде грампластинки.

ТЕРМИНАЛ — любое устройство для связи человека с компьютером. Слово это в переводе с английского значит — конечный. Так терминал называют потому, что стоит он как бы «на конце» вычислительной машины. Это ее уши, глаза и язык — средства ввода и вывода информации. Каких только терминалов не придумал для компьютера человек: печатающее устройство (принтер) — может отпечатать целую книгу, графопостроитель (плоттер) — нарисует любой чертеж, дисплей настоящий мультфильм на своем экране покажет. Терминалы видят, слышат и даже говорят с человеком. Персональный компьютер сам станет терминалом, если стоит он на конце линии связи с мощной, большой ЭВМ.



Хоть задач решит немало
За минуту ЭВМ,
Но, увы, без терминала,
Наш компьютер глух и нем.
Информации каналы,
Глаз, и ухо, и язык,
На вопросы терминала
Отвечают в тот же миг.

ДИСПЛЕЙ — электронное «лицо» компьютера. Самый удобный терминал для общения с компьютером. Внешне он выглядит как телевизор с клавиатурой от пишущей машинки. И компьютер и человек могут печатать и рисовать на экране дисплея. Так ведут они молчаливый разговор, легко понимая друг друга. То человек подскажет, то компьютер посоветует. В таком режиме диалога проще и быстрее решается любая сложная задача. Нарисовал конструктор на экране специальным электронным карандашом контуры будущего самолета, компьютер рассчитал для него подъемную силу и длину взлетной полосы... Изменил конструктор форму крыльев, компьютер мгновенно выдал на экран новые данные подъемной силы и взлетной полосы.



У тебя вопросов много?
Подскажу тебе я с кем
Ты в РЕЖИМЕ ДИАЛОГА
Сложных сто решишь проблем!
Отвечает без капризов,
Задавай вопрос быстрей.
Этот умный телевизор
Называется ДИСПЛЕЙ.

МИКРОПРОЦЕССОР — очень маленький, очень дешевый, но достаточно мощный и быстродействующий процессор. Электронная схема микропроцессора изготовлена на одном кристалле полупроводникового материала — кремния. Размеры микропроцессора не больше горошины, и похож он на маленького жучка с ножками-контактами. Такой маленький процессор можно вставить в любую машину и наделить ее, таким образом, машинным разумом. Благодаря специальным датчикам любая часть этой машины становится



Очень важен для прогресса
Всех машин и всех наук
Сын компьютера — процессор
ЭВМ послушный внук.
По размеру и по весу
Он сегодня очень мал,
И зовут — МИКРОПРОЦЕССОР
Этот кремния кристалл.

источником информации для микропроцессора. Этой же частью микропроцессор может управлять с помощью специальных исполнительных механизмов.

Телефонный аппарат и стиральная машина, автомобильный двигатель и кассовый аппарат — все эти машины стали гораздо умнее после того, как в них поместили микропроцессор.

ПРОГРАММА — описание решения задачи, заданное на языке вычислительной машины. Процесс переработки информации в компьютере полностью определяется программой, которую задает ему человек. В зависимости от этой программы универсальный компьютер становится умным советчиком врача, талантливым конструктором самолетов или опытным руководителем большого завода.

Удивительные таланты компьютера объясняются тем, что он способен очень быстро и четко выполнять несколько простейших арифметических и логических операций, последовательность которых и задает программа.

Программа должна быть четкой и понятной, как чертеж детали. Ясной и точной, как медицинский рецепт на изготовление лекарства. И безошибочной с точки зрения правил правописания компьютерного языка.

Профессия программиста, по образному выражению, пришла к нам из будущего.

```
720 INPUT " УКАЖИТЕ ИМЯ ФАЙЛА ДЛЯ ЧТЕНИЯ С ДИСКА"
725 OPEN FILE¤ FOR INPUT AS #1
730 INPUT #1,NMAX,NMIN
735 FOR K2=1 TO NMIN
740 INPUT #1,FAM¤(K2),TITLE¤(K2),VOL¤(K2),NUM¤(K2
742 INPUT #1,YEAR¤(K2),PUBL¤(K2)
760 NEXT K2
770 REM  ПРОКОНТРОЛИРОВАТЬ РЕЗУЛЬТАТЫ ЧТЕНИЯ С ДИ
775 CLOSE #1
780 RETURN
800 REM///
810 REM ===ПОДПРОГРАММА СОХРАНЕНИЯ БИБЛИОГРАФИИ Н
820 INPUT " УКАЖИТЕ ИМЯ ФАЙЛА ДЛЯ СОХРАНЕНИЯ НА Д
825 OPEN FILE¤ FOR OUTPUT AS #1
830 PRINT #1,NMAX,NMIN
835 FOR K2=1 TO NMIN
840 PRINT #1,FAM¤(K2),TITLE¤(K2),VOL¤(K2),NUM¤(K2
842 PRINT #1,YEAR¤(K2),PUBL¤(K2)
860 NEXT K2
870 REM  ПРОКОНТРОЛИРОВАТЬ РЕЗУЛЬТАТЫ ЗАПИСИ НА Д
875 CLOSE #1
880 RETURN
```

Как грудной малыш без мамы
Сам не может есть и пить,
Так компьютер без ПРОГРАММЫ
Шагу бы не мог ступить!
Я зря времени не трачу,
Как учитель на доске,
Я пишу ему задачу
На программном языке.
И ведет легко и быстро
Мой компьютер сложный счет,
И решенье программисту
На дисплее выдает.

В. Н. БУСЛЕНКО



Появление микро-ЭВМ позволило предложить новый подход к решению этой проблемы. За счет резкого снижения стоимости микропроцессорной техники недавно наблюдавшиеся концентрация и централизация вычислительных ресурсов уступают место децентрализации, проникновению вычислительной техники на рабочее место каждого потребителя. Приближение вычислительной техники к пользователю позволяет снизить расходы на передачу информации, повысить качество ее обработки и компьютерную грамотность. Таким образом, этот класс машин наиболее перспективен для использования на предприятиях, чему способствует минимальная их стоимость и небольшие габариты.

И. Н. ЛАДЫЧУК

«ИСКРА-555» В СИСТЕМЕ ОБРАБОТКИ ЭКОНОМИЧЕСКОЙ ИНФОРМАЦИИ

Как известно, существуют три основных способа обработки информации: децентрализованный, централизованный и распределенный. Если рассмотреть развитие вычислительной техники в динамике, то исторически первой возникла децентрализованная форма обработки информации, при которой техника устанавливается у конкретного пользователя и используется им для решения своих собственных задач. В период освоения машин первого поколения это была единственная форма обработки информации. Она имеет целый ряд достоинств.

Во-первых, предоставлялась возможность наиболее оперативной обработки документов, так как пользователь стал эффективно и оперативно решать свои задачи в любое удобное для него время. Кроме того, пропали потери времени на транспортировку или передачу данных от пользователя к вычислительной машине и обратно.

Во-вторых, снизилась стоимость аппаратных средств и программного обеспечения за счет разумной специализации. Конкретный пользователь никогда не сможет реализовать все многообразие возможностей, предоставляемых средствами вычислительной техники. Обычно круг задач более или менее четко определен, и пользователь может выбрать вычислительное средство, наиболее полно удовлетворяющее его потребностям.

В-третьих, повысилась надежность функционирования и живучесть системы. Высокая надежность функционирования обеспечивается за счет того, что подготовка данных, перенос на технические носители и обработка осуществляются непосредственно в местах возникновения информации. Теперь ошибки, допущенные при вводе информации, могут быть оперативно выявлены и скорректированы. Кроме того, выход из строя вычислительной установки у одного из пользователей не приведет к нарушению работы большого коллектива специалистов, как в случае организации многопользовательской работы от мощной единой ЭВМ.

Но по мере развития средств вычислительной техники, создания более производительных, обладающих большими возможностями вычислительных машин и систем все ярче начали вырисовываться и недостатки децентрализованной системы обработки информации.

Например, создание вычислительного центра на базе одной или нескольких ЭВМ требовало значительных затрат и было под силу только крупным предприятиям и организациям. Такие ВЦ требовали специального помещения, значительного штата обслуживающего персонала и немалых эксплуатационных расходов.

Особенно велики были затраты на разработку индивидуального программного обеспечения, так как каждый пользователь обычно разрабатывал свои программы самостоятельно. Отсюда в ряде случаев вытекала не

только высокая стоимость разработки программного обеспечения, но и низкая его эффективность, что также повышало стоимость обработки информации.

На ранних этапах развития немаловажную роль играл и субъективный фактор престижности. Пользователь часто приобретал более мощную и дорогостоящую машину, чем было действительно необходимо. Этому способствовала также ограниченная номенклатура выпускаемых ЭВМ и низкая «компьютерная грамотность».

Подытоживая сказанное, можно сделать вывод, что на ранних этапах развития средств вычислительной техники децентрализованная форма не могла служить основой для массового применения ЭВМ и была доступна только крупным организациям.

Все это привело к необходимости централизации средств вычислительной техники — создания вычислительных центров (ВЦ), которые одновременно обслуживали целый ряд пользователей. Создание вычислительных центров позволило повысить показатели экономической эффективности использования средств вычислительной техники и значительно расширить круг ее пользователей.

Стоимость обработки информации при централизованной форме обработки (ЦО) удалось снизить за счет ряда факторов.

Во-первых, за счет снижения капитальных затрат на строительство и эксплуатацию помещений для ЭВМ, так как средства вычислительной техники были сконцентрированы на одном ВЦ для целой группы предприятий.

Во-вторых, за счет уменьшения количества обслуживающего персонала. Одна мощная ЭВМ требует значительно меньше обслуживающего персонала, чем несколько менее мощных ЭВМ.

Отделение пользователей от непосредственного контакта с вычислительной техникой позволило углубить специализацию, за счет чего повысилось качество решаемых задач. Пользователь осуществлял квалифицированную постановку задач, а сотрудники ВЦ — разработку математического и программного обеспечения.

Централизация обработки позволила сократить количество пунктов обработки информации, укрупнить их и обеспечить более полную и равномерную загрузку, за счет чего снизилась стоимость обработки информации.

Однако наряду с положительными сторонами централизованная обработка имеет целый ряд недостатков, которые усугубляются по мере возрастания объемов вычислений.

Во-первых, отделение пользователя от вычислительных средств привело к необходимости качественной передачи информации от пользователя на ВЦ и обратно. На ранних этапах развития централизованной обработки такая передача обычно осуществлялась с помощью какого-либо транспорта. Причем пользователь представлял информацию на ВЦ в виде сводок, таблиц, ведомостей и т. п. На ВЦ вся информация переносилась на перфокарты и по каждому пользователю создавались массивы информации. При переносе информации с первичных документов на перфокарты не исключено появление ошибок. Такие ошибки очень трудно устраняются, так как работники ВЦ обычно недостаточно квалифицированно ориентируются в задачах пользователя, не могут проанализировать результаты расчетов с точки зрения пользователя. Пользователь же, передавая правильную исходную информацию и получая неправильную результативную, возвращает эту часть информации на повторную обработку или вынужден обрабатывать ее вручную. Поиск, выявление ошибок и пересчет — это довольно трудоемкие операции.

Входная информация, естественно, должна быть также определенным образом формализована, чтобы сотрудники ВЦ смогли ее обработать. Для подготовки исходной информации в форме, удобной для восприятия сотрудниками ВЦ, пользователю приходится осуществлять дополнительную ручную обработку исходной информации.

По мере развития технической базы средств вычислительной техники

совершенствовались операционные системы и внешние устройства ЭВМ. Это позволило использовать ЭВМ в так называемом режиме разделения времени. При работе в этом режиме у каждого пользователя устанавливается устройство связи с машиной — терминал, при помощи которого пользователь осуществляет обмен информации с машиной. Развитая операционная система автоматически осуществляет координацию работы пользователей, и у каждого из них создается иллюзия, что он работает с машиной один. Такая форма удобна тем, что исключаются промежуточные носители информации и повышается оперативность работы. Но повышение удобств достигается за счет увеличения стоимости обработки информации. Это происходит, во-первых, потому, что у каждого пользователя должно быть терминальное устройство связи с ЭВМ. Во-вторых, передача информации от терминала к ЭВМ и обратно должна осуществляться по каналам связи. Причем для существующих ЭВМ используются электрические каналы связи, представляющие собой наиболее дорогостоящую часть вычислительной системы.

Стоимость использования систем зависит, во-первых, от стоимости передачи информации по каналам связи, во-вторых, от стоимости обработки информации процессорами. Как показывают экономические расчеты и анализ тенденций снижения средств передачи и обработки данных, наибольшее влияние на стоимость системы оказывают затраты на аренду каналов. Как стоимость обработки информации, так и стоимость ее передачи падают, однако снижение стоимости обработки информации происходит гораздо быстрее, чем стоимость передачи. Поэтому в ближайшем будущем именно стоимость передачи информации будет определять суммарную стоимость пользования вычислительными системами.

При централизованной обработке информации у пользователя отсутствует какая-либо вычислительная техника, поэтому вся информация должна передаваться на вычислительный центр и обратно. При обработке экономической информации операции по ее обработке самые простые и обычно не выходят за пределы четырех арифметических действий. Но объемы ее весьма значительны даже для средних и мелких предприятий. Это приводит к передаче огромных потоков информации по каналам связи, что резко повышает требования к средствам передачи данных.

Таким образом, при централизованной обработке информации возникает проблема эффективной системы передачи данных от пользователя к ЭВМ и обратно.

Поэтому централизованная форма работы наиболее эффективна в том случае, когда сложность обработки информации высокая, а объем данных, передаваемых по каналам связи, относительно невелик.

В последнее время все большее распространение получает распределенная обработка информации. Организуется она на сетях ЭВМ или однородных вычислительных системах. Однородные вычислительные системы позволяют принципиально достичь любого быстродействия за счет распараллеливания операций обработки информации. Однако они, несмотря на все преимущества, являются специализированными и предназначены для решения определенного круга задач, так как не любые задачи допускают их параллельное решение. Для массовой же вычислительной техники вовсе не обязательно достижение максимально большой производительности, более того, в последнее время для массового пользователя сдерживающим фактором является не быстродействие процессора, а быстродействие устройств ввода-вывода.

В настоящее время мощности и возможностей отдельных микро-ЭВМ вполне достаточно для обработки экономической информации мелких и средних предприятий. Такая обработка носит в основном рутинный характер и вполне доступна серийным микро-ЭВМ. Поэтому такие машины находят все большее применение для ведения учетно-плановых работ на малых и средних предприятиях, причем 95% их

времени используется для решения учетных задач. Эти микро-ЭВМ за рубежом стоят 3—5 тыс. долл. Имеется достаточно большой выбор пакетов прикладных программ (ППП) для ведения бухгалтерского учета на микро-ЭВМ. Так, созданный в Англии ППП «МАП компьютер системз» отвечает требованиям большинства фирм. Другой ППП для начисления заработной платы на микро-ЭВМ фирмы «Omicron» может быть применен на предприятиях с общей численностью работников до 9999. При этом осуществляется 30 начислений в минуту.

В наиболее развитых капиталистических странах уже в середине 70-х годов предусматривалось, что значительное число мелких предприятий будет использовать в перспективе для ведения учетно-плановых и аналитических работ микро-ЭВМ. В западноевропейских странах, например, насчитывалось 500 тыс. предприятий и организаций с числом занятых от 20 до 400 человек, а в США — 600 тыс. фирм с числом занятых от 10 до 250 человек и 3 млн. фирм с числом занятых менее 10 человек. Предполагалось, что каждое предприятие с числом занятых свыше 10 человек будет использовать микро-ЭВМ. В 1983 г., например в Англии, уже насчитывалось около 2 млн. микро-ЭВМ, из которых на долю машин, применяемых в экономическом управлении предприятиями и организациями, приходилось около 50%. Таким образом, произошло значительное насыщение микромашинами мелких и средних предприятий.

Для обработки экономической информации целесообразно использовать специализированные микро-ЭВМ, ориентированные на решение определенного класса задач. Такие специализированные микро-ЭВМ, ориентированные на обработку экономической информации, получили название электронно-бухгалтерских машин (ЭБМ). Отличие ЭБМ от универсальных микро-ЭВМ заключается в проблемно-ориентированном программном обеспечении, упрощенной системе команд и соответствующем наборе периферийного оборудования.

Первая отечественная машина такого класса — «Искра-554», на смену ей пришла более современная ЭБМ — «Искра-555», выпускающаяся в настоящее время.

Проблемно-ориентированная ЭБМ «Искра-555» предназначена для механизированного и автоматизированного решения бухгалтерских, плановых, учетно-статистических и других экономических задач с одновременным формированием многострочных и многографных документов различной сложности.

Областями применения ЭБМ «Искра-555» являются бухгалтерии, плановые и диспетчерские службы промышленных, торговых, транспортных, жилищно-коммунальных и других предприятий и организаций, сберкассы, склады, базы материального снабжения и сбыта, машинно-счетные станции и бюро, АСУ различной ориентации.

Операционная система ЭБМ «Искра-555», включающая транслятор, ретранслятор и отладочные средства, реализована на микропрограммном уровне и хранится в постоянном запоминающем устройстве.

Выход в канал связи позволяет использовать ЭБМ как развитой терминал машин ЕС ЭВМ и СМ ЭВМ.

Дисплей, имеющий два формата отображения, обеспечивает эффективное взаимодействие пользователя с машиной как в процессе обработки данных, так и при отладке программ. Индикатор контроля и настройки позволяет быстро локализовать и устранить неисправность в процессе ремонта и технического обслуживания на месте установки ЭБМ.

Технические характеристики ЭБМ „Искра-555“

Разрядность вводимой и обрабатываемой информации:	
цифровая	— до 16
алфавитно-цифровая	— до 256

Количество адресуемых в программе регистров:	
индексные	— до 4096
числовые	— до 4096×П
алфавитные	— до 4096×П (П=0÷1025 — количество массивов, задаваемых в программе)
Точность вычислений, задаваемая в программе	— до 7 знаков после запятой
Язык программирования	— символьный проблемно-ориентированный язык ЯМБ
Элементная база процессора	— микропроцессорный набор серии К589
Быстродействие процессора (на уровне операций типа регистр-регистр), оп/с	650 000
Оперативное запоминающее устройство (ОЗУ):	
объем, Кбайт	— 16÷32
Алфавитно-цифровое печатающее устройство (АЦПУ):	
тип ПУ	— „Даро-1156"
тип печати	— матричный
скорость печати (печать в двух направлениях), зн/с	— 100
длина бумагоопорного вала, мм	— 470
соотношение длин частей разрезного бумагоопорного вала	— 1:2 или 2:1
количество печатных знаков в строке	— 178
количество печатаемых экземпляров	— 1÷5
виды бумажных носителей	— бумага рулонная перфорированная и неперфорированная, бланки
закладка бланков	— передняя и задняя
Устройство ввода-вывода с перфоносителя (УВП):	
скорость ввода на перфоленту, зн/с	— 20 или 75
скорость ввода с перфоленты, зн/с	— 200 или 300
Устройство записи-считывания с магнитной карты (УЗСМК):	
емкость магнитной карты, байт	— 512
Накопитель на магнитной ленте в мини-кассете (КНМЛ):	
а) тип I:	
метод записи	— CRB (импульсный)
совместимость с СМ ЭВМ	— нет
объем информации на одной магнитной ленте в мини-кассете, Кбайт	— 50
б) тип II:	
метод записи	— PhE (фазовый)
совместимость с СМ ЭВМ	— есть (для СМ-3, СМ-4 только при комплектации устройством управления СМ-5211)
количество дорожек	— 2
объем информации на одной дорожке, Кбайт	— 200
Накопитель на миниатюрном гибком магнитном диске (НМГМД):	
тип накопителя	— ЕС-5088
оперативно доступная емкость на один дисковод, Кбайт	— 80

Накопитель на гибком магнитном диске (НГМД):	
тип накопителя	— PL×45D2 или ЕС-5074
оперативно-доступная емкость на один дисковод, Кбайт	— 250
Накопитель на магнитном диске (НМД):	
тип накопителя	— Р414М, „Изот-1370", СМ-5400
оперативно доступная емкость, Мбайт	— 5,0 для „Изот-1370" и СМ-5400
	— 1,3 для Р414М
Накопитель на магнитной ленте (НМЛ):	
тип накопителя	— СМ-5300.01
объем информации на одной магнитной ленте, Мбайт	— 6÷9 (в зависимости от длины зоны)
Кодирование данных на:	
магнитной ленте и магнитных дисках	— КОИ-7 (ГОСТ 13051—74)
перфоленте	— КОИ-7 (ГОСТ 13052—74) или МТК-2
Дисплей:	
тип	— символьный на ЭЛТ
емкость экрана:	
малый формат знаков	— 16 строк × 64 знака
большой формат знаков	— 8 строк × 32 знака
Телекоммуникационные средства ЭВМ	— блок интерфейсный БИ АПД: 1 канал по стыку С2 — многоканальный процессор теледоступа: 8 каналов по стыку ИРПС, 2 канала по стыку С2
Параметры телекоммуникационных интерфейсов:	
а) стык С2:	
канал электросвязи	— выделенный телефонный с 2-проводным окончанием, физические пары
возможные модемы	— модемы типа ЕС-8001, ЕС-8005, ЕС-8027
протокол дистанционного взаимодействия	— асинхронный, абонентских пунктов ЕС АП-70 и АП-1 для выделенного канала с многоточечным включением
скорость передачи данных, бит/с	— 200, 600, 1200, 2400, 4800, 9600
б) стык ИРПС („токовая петля"):	
канал электросвязи	— физические пары
протокол дистанционного взаимодействия	— асинхронный протокол ЭБТ „Нева-501", (совместимый с СМ-1300, СМ-3, СМ-4)
скорость передачи данных, бит/с	— до 9600
дальность передачи при скорости 9600 бит/с, м	— 500
Габаритные размеры стола оператора, мм	— 1000×780×770
Электропитание ЭВМ	— однофазная сеть, $220В^{+10\%}_{-15\%}$ с частотой 50 Гц

Потребляемая мощность, ВА	— 650÷1650 (в зависимости от исполнения)
Стоимость ЭБМ, тыс. руб.	— 15÷35 (в зависимости от исполнения)

«Искра-555» выпускается в различных исполнениях, образующих ряд машин, которые отличаются набором устройств ввода-вывода, емкостью оперативного запоминающего устройства и стоимостью. Это позволяет получать максимальный экономический эффект от применения ЭБМ, используя соответствующие ее исполнения для решения определенного вида задач. Некоторые исполнения машины являются функциональными аналогами и отличаются лишь номенклатурой примененных в них устройств ввода-вывода.

Специфика применения ЭБМ накладывает определенные требования, которым должны отвечать машины данного класса, а именно: простота и доступность в эксплуатации, низкая стоимость и возможность частой смены относительно простых задач, т. е. простота разработки прикладного обеспечения. Эти критерии определили как структуру самих машин, так и язык программирования для ЭБМ.

Для отечественных ЭБМ был разработан специальный проблемно-ориентированный язык для решения задач оперативной работы с данными и документами. Язык получил название ЯМБ (язык машин бухгалтерских). Структура языка ЯМБ позволяет производить любые операции по обработке данных, аналогичные универсальным ЭВМ, управлять устройствами ввода-вывода, составлять описание форм выходных документов и массивов данных. Язык дает возможность строить разветвленные сегментированные программы обработки цифровой и алфавитно-цифровой информации. Русская мнемоника языка и простота программирования позволяют оперировать с ним специалистам с минимальной подготовкой.

Кроме «Искры-555», был разработан настольный проблемно-ориентированный «интеллектуальный» терминал ЭБТ «Нева-501», предназначенный для решения оперативных бухгалтерских, плановых и других задач первичной обработки экономической информации с одновременным оформлением документов.

Терминал обеспечивает автономное использование и работу в системах с обменом данными технических носителей и с обменом по каналам связи. Области применения — учетные, плановые, бухгалтерские и другие экономические службы организаций и предприятий промышленности, торговли, в сельском хозяйстве, на транспорте, а также в системах ЦСУ, Госбанка и других формах народного хозяйства.

Выпускается ЭБТ в различных исполнениях, образующих ряд «интеллектуальных» терминалов, которые отличаются набором устройств ввода-вывода и емкостью ОЗУ.

Телекоммуникационные средства позволяют использовать ЭБТ в качестве развитого терминала для машин ЕС и СМ ЭВМ. Кроме того, эти средства обеспечивают дистанционное подключение ЭБТ и ЭБМ «Искра-555» в качестве центральной машины многотерминального комплекса. Предоставляется возможность организации дистанционного взаимодействия терминалов между собой.

Совместимость ЭБТ «Нева-501» с ЕС ЭВМ по перфоленте, гибкому магнитному диску, магнитной ленте в миникассете и каналу связи обеспечивает возможность системного использования ЭБТ. Совместимость по каналу связи и техническим носителям с ЭБМ «Искра-555» позволяет создавать локальные системы обработки данных на базе только указанных средств.

Широкому внедрению ЭБМ в производство препятствует отсутствие специального программного обеспечения для решения задач обработки экономической информации. Мелкие и средние предприятия, для которых в основном предназначены ЭБМ, не в состоянии самостоятельно разработать комплекс программ в связи с отсутствием специа-

листов и высокой стоимостью разработки программного обеспечения.

На кафедре экономической кибернетики Днепропетровского сельскохозяйственного института на базе ЭБМ «Искра-555» разработана локальная система обработки учетной информации для автотранспортных предприятий АПК. Разработанная система позволяет в автоматическом режиме выдавать целый ряд сводных и аналитических ведомостей и осуществлять ответ на запросы.

В связи с малым объемом памяти оперативного запоминающего устройства ОЗУ ЭБМ «Искра-555» — 32 Кбайта, информацию по мере поступления и обработки необходимо размещать на внешних быстродействующих носителях — магнитных дисках. Ежедневная запись информации осуществляется на гибкий магнитный диск с последующей перезаписью на жесткий. Это позволяет значительно повысить качество, надежность и достоверность обработки информации. При записи на ГМД осуществляется контроль правильности вводимой информации и при необходимости ее корректировка. Перезапись на жесткий диск, по мере заполнения ГМД, осуществляется после проверки и корректировки. Это позволяет свести к минимуму количество обращений к жесткому диску и тем самым снизить вероятность неправильной записи информации, ее порчи.

В соответствии с поставленными задачами для системы обработки информации разработан ряд программных модулей, основными из которых являются модуль программы формирования нормативно-справочной информации и модули программ для обработки и формирования соответствующих машинограмм. Все программные модули хранятся на гибком магнитном диске и при работе переписываются в ОЗУ. Модульное программное обеспечение комплекса задач занимает около 22% объема одного гибкого диска. Программное обеспечение может расширяться и обновляться для решения более сложных задач учета изменений в системе начисления заработной платы.

Разработанное информационное и программное обеспечение позволяет осуществить обработку информации по следующим комплексам задач: учет труда и заработной платы, оперативный учет технико-эксплуатационных показателей, учет товарно-материальных ценностей, учет реализации выполненных услуг и целый ряд других задач.

По каждому участку учета разработана информационная модель и соответствующее программное обеспечение. Например, в содержание модели по учету труда и заработной платы входит информация учета численности работающих по основным категориям; учета выполнения норм выработки и производительности труда; учета фактического (начисленного) фонда заработной платы по основным категориям; учета расчетов с органами соцстраха, финансовыми организациями, депонентами; учета других операций, связанных с заработной платой, — расчеты по отпускам, временной нетрудоспособности, расчеты по премиям, надбавкам и пр.

При разработке систем обработки экономической информации в нижних звеньях агропромышленного комплекса на уровне отдельных предприятий возникает ряд проблем. Как показали проведенные исследования, подавляющее большинство экономической информации (свыше 90%) составляет учетная информация. В связи с ограниченными возможностями ЭБМ «Искра-555» (малый объём ОЗУ, невысокое быстродействие) и отсутствием для них стандартных СУБД для решения учетных задач в нижних звеньях агропромышленного комплекса имеет смысл создавать локальные базы данных по отдельным задачам с разумным использованием и дублированием информации. Создание единого банка данных целесообразно в информационно-поисковых многоуровневых системах, где часто возникают ситуации, требующие различных нестандартных запросов. В случае же обработки первичной учетной информации, где обработка осуществляется по типовым алгоритмам, весьма значительные затраты на разработку соответствующего программного обес-

печения и поддержание банка данных на актуальном уровне не оправдывают себя.

При создании систем обработки учетной информации в нижних звеньях агропромышленного комплекса с использованием ЭБМ имеются три «узких» места, которые необходимо преодолеть программным или организационным путем. Первое — это огромный объем первичной информации и низкая скорость ввода ее в ЭБМ. Например, только при решении комплекса задач по учету труда и заработной платы объем путевых листов для мелких и средних предприятий превышает 100 тыс. в год, каждый из которых содержит около 250 знаков. Эта трудность преодолевается созданием сети звездообразной структуры с использованием в качестве центральной ЭБМ «Искра-555» соответствующего исполнения, а в качестве терминальных — «Нева-501». В этом случае ЭБТ «Нева-501» устанавливаются непосредственно в местах сосредоточения информации, и их количеством можно регулировать пропускную способность сети по входу. Связь отдельных ЭБМ и ЭБТ осуществляется с использованием интерфейса ИРПС. Это позволяет подключать до восьми терминалов ЭБТ «Нева-501», находящихся на расстоянии до 500 м, что полностью удовлетворяет запросы пользователя. При вводе информации с клавиатуры ЭБТ осуществляется ее предварительная обработка и запись на гибкий диск. На центральную ЭБМ передается уже обработанная информация. В случае выхода из строя линии связи возможна передача информации непосредственно на техническом носителе — гибком диске.

На стадии обработки экономической информации большую долю занимают различные сортировки первичных массивов в разрезе определенных показателей. Как показали проведенные исследования, в связи с низким быстродействием ЭБМ «Искра-555» обработка месячных массивов информации путем соответствующих сортировок даже для мелких предприятий занимает недопустимо большое время, что не позволяет уложиться в заданные сроки выпуска результативной информации. Выходом из положения является первичная обработка массивов информации при ее вводе на ЭБТ и создание унифицированных структур записей по отдельным учетным задачам. Такой подход позволяет сортировку заменить простым алгебраическим суммированием записей в определенных регистрах. В случае невозможности создания унифицированных массивов с фиксированными адресами создаются специальные справочники признаков, и вместо сортировки массива осуществляется сортировка справочника.

К третьему «узкому» месту относится вывод информации на печать. Для большинства учетных задач необходим не только визуальный контроль и восприятие информации с экрана дисплея, но и материальный документ. В связи с невысоким быстродействием печатающего устройства «Даро-1156» возникают сложности со сроками вывода информации на печать по таким громоздким задачам, как, например, учет товарно-материальных ценностей и т. д. В этом случае необходимо предусмотреть минимум выходных форм при максимальной их содержательности или вывод информации на технический носитель и передача на СМ и ЕС ЭВМ для последующей распечатки на быстродействующих АЦПУ.

Система предназначена для применения на мелких и средних предприятиях. Максимальный объем обрабатываемой информации лимитируется типом накопителя на жестком магнитном диске центральной ЭБМ «Искра-555». При использовании накопителя типа Р414М возможна обработка информации на предприятиях с общей численностью работающих до 650 человек, а при использовании накопителей типа «Изот-1370» или «СМ-5400» — до 2500 человек.

Дальнейшее развитие систем обработки экономической информации связано с совершенствованием персональных ЭВМ (ПЭВМ). В частности, для ЭВМ серии «Искра» планируется выполнить следующие работы:

изменить параметры устройств вне-

шней памяти (увеличение емкости, уменьшение габаритов, повышение надежности работы);

улучшить средства диалогового взаимодействия (включение в состав машины цветного дисплея, устройств речевого ввода-вывода, манипулятора типа «мышь» и т. д.);

создать новые ранги связи и протоколы обмена (различные типы локальных сетей);

увеличить профессиональную ориентацию программных средств и т. д.

Новая модификация получила название «Искра 1030.11». Она выполнена в виде компактного основного модуля размером 480×420×180 мм, в котором находятся электронные блоки, блок питания и два накопителя на гибких магнитных дисках. К основному модулю подключаются дисплей, устанавливаемый либо на верхней панели модуля, либо рядом с ней, клавиатура, свободно перемещаемая пользователем, и алфавитно-цифровое печатающее устройство.

Дисплей ПЭВМ имеет экран с диагональю 40 см и позволяет выводить алфавитно-цифровую информацию в двух форматах (25 строк по 80 символов и 25 строк по 40 символов) и графическую информацию с высокой и средней разрешающей способностью (640×200 точек и 320×200 точек).

Алфавитно-цифровое печатающее устройство типа К6312М позволяет выводить на рулон полный набор алфавитно-цифровых символов, специальных знаков и символов псевдографики, задавать по программе различные типы шрифтов для печати, выполнять автоматическую и ручную подачу бумаги, получать на печати точную копию графического экрана. Ширина печатного вала — 420 мм. Скорость печати — 100 знаков/с. Число знаков в строке — 132, 163 или 233.

Кроме того, может использоваться печатающее устройство типа «Искра 041ША», обеспечивающее вывод алфавитно-цифровых символов и специальных знаков в соответствии с клавиатурой, работу с разрезным валом для одновременного оформления двух документов, обработку бланков различной формы с возможностью одновременного вывода на бланк и на рулон, автоматическую протяжку и построчное перемещение, перемещение бланков по программе в процессе печати. Ширина вала — 420 мм. Скорость печати — 150 знаков/с. Число знаков в строке — 167.

Программное обеспечение ПЭВМ включает модульную операционную систему АДОС, хранящуюся на гибком мини-диске. Языки программирования, поддерживаемые этой операционной системой: БЕЙСИК А — интерпретируемый вариант с операторами обработки графической информации и макроассемблер MASM с возможностью редактирования, отладки и компоновки программ.

Предусматривается включение в состав языков программирования языков ПАСКАЛЬ и СИ. Кроме перечисленных языков, в ПЭВМ «Искра 1030.11» реализована расширенная версия языка ЯМБ, которым оснащены ЭБМ «Искра-555» и ЭБТ «Нева-501». Это позволяет использовать большой объем пакетов прикладных программ, разработанных ранее для данных машин.

Литература

Заворотный В. И. Программирование на языке ЯМБ. — М.: Финансы и статистика, 1984. — 160 с.

Ладычук И. Н. Учет труда и заработной платы с использованием ЭБМ «Искра-555». — М.: Финансы и статистика, 1986. — 111 с.

Ладычук И. Н., Сорочинская Л. А. Автоматизация учета на автотранспортных предприятиях // Планирование и учет в сельскохозяйственных предприятиях. — 1985. — № 4.

Ладычук И. Н. Автоматизация учета труда и заработной платы на автотранспортных предприятиях АПК // Планирование и учет в сельскохозяйственных предприятиях. — 1986. — № 4.

Лукин В. В. Обработка данных бухгалтерского учета на микро-ЭВМ // Бухгалтерский учет. — 1984. — № 9.

Стецюра Г. Г. Организация обмена информацией в микропроцессорных системах. — М.: Машиностроение, 1982. — 56 с.

Ярошевская М. Ю., Беручка Ю. И., Кейлин А. Я. Электронная бухгалтерская машина «Искра-555»: эксплуатация и программирование. — М.: Финансы и статистика, 1983. — 192 с.

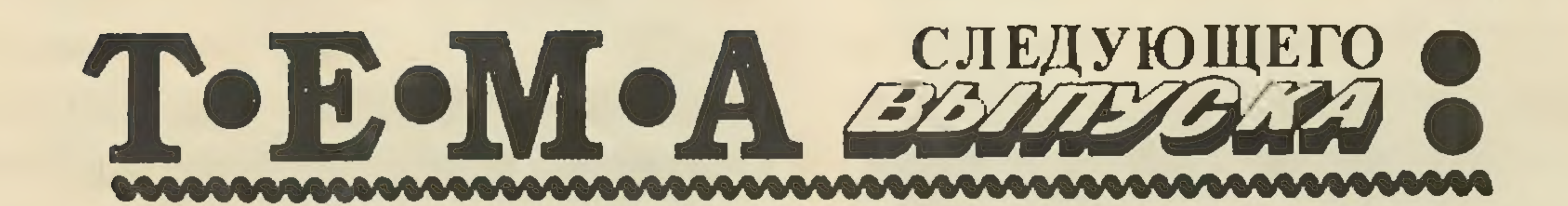
Ярошевская М. Б. Персональная ЭВМ «Искра 1030.11» // Микропроцессорные средства и системы. — 1986. — № 4.

Л 69 Логическое программирование. — М.: Знание, 1988. — 48 с. — (Новое в жизни, науке, технике. Сер. «Вычислительная техника и ее применение»; № 9).
15 к.

Авторы на основе современных достижений обсуждают самые острые проблемы развития вычислительной техники: новые архитектуры ЭВМ, способы повышения производительности ЭВМ, перспективные системы программного обеспечения.

Брошюра рассчитана на широкий круг читателей, интересующихся перспективными разработками вычислительной техники.

2404090000 ББК 32.973—01



В ОКЕАНЕ ДАННЫХ

Научно-популярное издание

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Гл. отраслевой редактор *Л. А. Ерлыкин*
Редактор *Б. М. Васильев*
Мл. редактор *Н. А. Васильева*
Художники *В. Н. Конюхов, К. Н. Мошкин*
Худож. редактор *М. А. Гусева*
Техн. редактор *Т. В. Луговская*
Корректор *В. И. Гуляева*

ИБ № 9599

Сдано в набор 24.05.88. Подписано к печати 21.07.88. Т-05423. Формат бумаги $70 \times 100^1/_{16}$. Бумага офсетная. Гарнитура Гельветика. Печать офсетная. Усл. печ. л. 3,90. Усл. кр.-отт. 8,45. Уч.-изд. л. 4,38. Тираж 65 820 экз. Заказ 2163. Цена 15 коп. Издательство «Знание». 101835, ГСП, Москва, Центр, проезд Серова, д. 4. Индекс заказа 884709.
Ордена Трудового Красного Знамени Калининский полиграфический комбинат Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 170024, г. Калинин, пр. Ленина, 5.

Цена 15 коп. Индекс 70195

Адрес подписчика:



Подписная научно-популярная серия

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЕ ПРИМЕНЕНИЕ

Дорогой читатель!

Брошюры этой серии в розничную продажу не поступают, поэтому своевременно оформляйте подписку.
Подписка на брошюры издательства «Знание» ежеквартальная, принимается в любом отделении «Союзпечати».

Наш адрес: Москва, Центр, проезд Серова, д. 4